1.Odd or Even Number

```
#include <stdio.h>
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num % 2 == 0)
        printf("%d is Even\n", num);
    else
        printf("%d is Odd\n", num);
    return 0;
}
```

Output:

```
Enter a number: 4
4 is Even
```

---

2. Fibonacci using Recursion

```
#include <stdio.h>
int fib(int n) {
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}
int main() {
    int i, n = 5;
    for (i = 0; i < n; i++)
        printf("%d ", fib(i));
    return 0;
}
```

Output:

CopyEdit

```
0 1 1 2 3
```

---

3. Array Insert and Display

```
#include <stdio.h>
int main() {
    int arr[100], n = 3, i;
    arr[0] = 10; arr[1] = 20; arr[2] = 30;
    arr[n++] = 40; // Insert
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Output:

CopyEdit

```
10 20 30 40
```

---

4. Voting Eligibility

```
#include <stdio.h>
int main() {
    int age;
    printf("Enter age: ");
    scanf("%d", &age);
    if (age >= 18)
        printf("Eligible to vote\n");
    else
        printf("Not eligible to vote\n");
    return 0;
}
```**Output:**

```
Enter age: 17

Not eligible to vote
```

5. Binary Search

```
#include <stdio.h>
int main() {
    int a[] = {10,20,30,40}, k = 30, l = 0, h = 3, m;
    while(l<=h) {
        m=(l+h)/2;
        if(a[m]==k) return printf("Found\n"),0;
        k>a[m] ? (l=m+1) : (h=m-1);
    }
    printf("Not found\n");
}
```

Output:

```
Found at index 2
```

6. Matrix Addition 3x3

```
#include <stdio.h>
int main() {
    int a[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int b[3][3] = {{9,8,7},{6,5,4},{3,2,1}};
    int c[3][3], i, j;
    for (i=0;i<3;i++) {
        for (j=0;j<3;j++) {
            c[i][j] = a[i][j] + b[i][j];
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
CopyEdit
```

```
10 10 10
10 10 10
10 10 10
```

---

7. Display Array Elements

```
#include <stdio.h>
int main() {
    int a[] = {10, 20, 30, 40, 50}, i;
    for (i = 0; i < 5; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
CopyEdit
10 20 30 40 50
```

---

8. Multiplication Table of 5

```
#include <stdio.h>
int main() {
    int i;
    for (i = 1; i <= 10; i++)
        printf("5 x %d = %d\n", i, 5*i);
    return 0;
}
```

Output:

```
5 x 1 = 5
...
5 x 10 = 50
```

---

9. Stack for Postfix Expression

```
#include <stdio.h>
#include <ctype.h>
int stack[100], top = -1;
void push(int x) { stack[++top] = x; }
int pop() { return stack[top--]; }
int main() {
    char exp[] = "23*5+";
    int i, op1, op2;
    for (i = 0; exp[i]; i++) {
        if (isdigit(exp[i]))
            push(exp[i] - '0');
        else {
            op2 = pop(); op1 = pop();
            switch (exp[i]) {
                case '+': push(op1 + op2); break;
                case '*': push(op1 * op2); break;
            }
        }
    }
    printf("Result = %d\n", pop());
    return 0;
}
```

Output:

```
Result = 11
```

---

10. Greatest of Three Numbers

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

```c
int main() {
    int a=10, b=20, c=15;
    if (a > b && a > c)
        printf("%d is greatest\n", a);
    else if (b > c)
        printf("%d is greatest\n", b);
    else
        printf("%d is greatest\n", c);
    return 0;
}
```

Output:

```
20 is greatest
```

12. Sort Array Ascending

```c
#include <stdio.h>
int main() {
    int a[] = {30, 10, 40, 20}, i, j, temp;
    for (i = 0; i < 4; i++)
        for (j = i+1; j < 4; j++)
            if (a[i] > a[j]) {
                temp = a[i]; a[i] = a[j]; a[j] = temp;
            }
    for (i = 0; i < 4; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
10 20 30 40
```

---

13. Factorial Using Recursion

```c
#include <stdio.h>
int fact(int n) {
    if (n == 0) return 1;
    return n * fact(n - 1);
}
int main() {
    int n = 5;
    printf("Factorial of %d is %d\n", n, fact(n));
    return 0;
}
```

Output:

```
Factorial of 5 is 120
```

---

14. Linked List Insert at Beginning

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
void display(struct Node* head) {
    while (head) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}
int main() {
    struct Node* head = NULL;
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = 10; newNode->next = head; head = newNode;

    newNode = malloc(sizeof(struct Node));
    newNode->data = 20; newNode->next = head; head = newNode;
```

```
    display(head);
    return 0;
}
```

Output:

```
rust
CopyEdit
20 -> 10 -> NULL
```

---

15. Linear Search

```
#include <stdio.h>
int main() {
    int a[] = {5, 10, 15, 20}, i, key = 15;
    for (i = 0; i < 4; i++) {
        if (a[i] == key) {
            printf("Found at index %d\n", i);
            return 0;
        }
    }
    printf("Not found\n");
    return 0;
}
```

Output:

```
Found at index 2
```

---

16. Tree Inorder Traversal

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
```

```
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* newNode(int data) {
    struct Node* node = malloc(sizeof(struct Node));
    node->data = data; node->left = node->right = NULL;
    return node;
}
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
int main() {
    struct Node* root = newNode(10);
    root->left = newNode(5);
    root->right = newNode(15);
    inorder(root);
    return 0;
}
```

Output:

```
5 10 15
```

17.Bubble sort

```
#include <stdio.h>

int main() {
    int arr[100], n, i, j, temp;

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

    printf("Enter %d elements:\n", n);
```

```
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - 1 - i; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }

    printf("Sorted array:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

18.Selection sort

```
#include <stdio.h>

int main() {
    int arr[100], n, i, j, min, temp;

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

    printf("Enter %d elements:\n", 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]);
    }

    return 0;
}
```

19.Insertion sort

```
#include <stdio.h>

int main() {
    int arr[100], n, i, j, key;

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

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

    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
```

```
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }

    printf("Sorted array:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

20.seperate chaining

```
#include <stdio.h>

#define TABLE_SIZE 5
#define MAX_CHAIN 3  // Max items per bucket (simulate chaining)

// Hash table using a 2D array
int hashTable[TABLE_SIZE][MAX_CHAIN];
int count[TABLE_SIZE];  // To track number of items in each bucket

// Hash function
int hash(int key) {
    return key % TABLE_SIZE;
}

// Insert a key
void insert(int key) {
    int index = hash(key);

    if (count[index] < MAX_CHAIN) {
        hashTable[index][count[index]] = key;
        count[index]++;
    } else {
```

```
        printf("Bucket %d is full. Cannot insert %d.\n", index, key);
    }
}

// Display the hash table
void display() {
    for (int i = 0; i < TABLE_SIZE; i++) {
        printf("Index %d: ", i);
        for (int j = 0; j < count[i]; j++) {
            printf("%d ", hashTable[i][j]);
        }
        printf("\n");
    }
}

// Main function
int main() {
    // Initialize count array
    for (int i = 0; i < TABLE_SIZE; i++) {
        count[i] = 0;
    }

    // Insert some keys
    insert(10);
    insert(15);
    insert(7);
    insert(22);
    insert(5);
    insert(30);  // May exceed bucket size

    // Display the table
    display();

    return 0;
}
```

21.Open addressing

```c
#include <stdio.h>

#define TABLE_SIZE 10
#define EMPTY -1

int hashTable[TABLE_SIZE];

// Hash function
int hash(int key) {
    return key % TABLE_SIZE;
}

// Insert using linear probing
void insert(int key) {
    int index = hash(key);

    // Try to find an empty slot
    for (int i = 0; i < TABLE_SIZE; i++) {
        int try = (index + i) % TABLE_SIZE;
        if (hashTable[try] == EMPTY) {
            hashTable[try] = key;
            return;
        }
    }

    printf("Hash table is full. Cannot insert %d\n", key);
}

// Display the hash table
void display() {
    printf("Hash Table:\n");
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i] != EMPTY)
            printf("Index %d: %d\n", i, hashTable[i]);
        else
            printf("Index %d: EMPTY\n", i);
    }
}
```

```
// Main function
int main() {
    // Initialize table with EMPTY
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i] = EMPTY;
    }

    // Insert some keys
    insert(10);
    insert(20);
    insert(30);  // Collides with 10
    insert(25);
    insert(15);  // Collides with 25 and 15

    display();

    return 0;
}
```

22.Hashing

```
#include <stdio.h>

#define TABLE_SIZE 10
#define EMPTY -1

int hashTable[TABLE_SIZE];

// Hash function
int hash(int key) {
    return key % TABLE_SIZE;
}

// Insert function with linear probing
void insert(int key) {
    int index = hash(key);
```

```
    // Try to find an empty spot
    for (int i = 0; i < TABLE_SIZE; i++) {
        int newIndex = (index + i) % TABLE_SIZE;
        if (hashTable[newIndex] == EMPTY) {
            hashTable[newIndex] = key;
            printf("Inserted %d at index %d\n", key, newIndex);
            return;
        }
    }

    printf("Table is full. Couldn't insert %d\n", key);
}

// Search function
void search(int key) {
    int index = hash(key);

    for (int i = 0; i < TABLE_SIZE; i++) {
        int newIndex = (index + i) % TABLE_SIZE;
        if (hashTable[newIndex] == key) {
            printf("Found %d at index %d\n", key, newIndex);
            return;
        }
        if (hashTable[newIndex] == EMPTY) {
            break; // Stop early if empty slot found
        }
    }

    printf("Key %d not found\n", key);
}

// Display the hash table
void display() {
    printf("\nHash Table:\n");
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i] != EMPTY)
            printf("Index %d: %d\n", i, hashTable[i]);
        else
```

```
            printf("Index %d: EMPTY\n", i);
    }
    printf("\n");
}

// Main function
int main() {
    // Initialize table with EMPTY values
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i] = EMPTY;
    }

    // Insert some values
    insert(10);
    insert(20);
    insert(30);
    insert(25);
    insert(35);
    insert(15);

    // Display the table
    display();

    // Search for values
    search(25);
    search(99);

    return 0;
}
```