

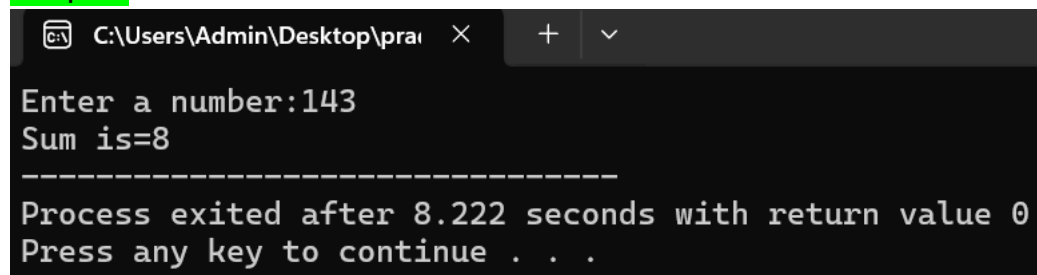
1. Write a program to find the sum of digits.

Program:

```
#include<stdio.h>

int main()
{
    int n,sum=0,m;
    printf("Enter a number:");
    scanf("%d",&n);
    while(n>0)
    {
        m=n%10;
        sum=sum+m;
        n=n/10;
    }
    printf("Sum is=%d",sum);
    return 0;
}
```

Output:

A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Users\Admin\Desktop\prai'. The window contains the following text: 'Enter a number:143', 'Sum is=8', a separator line of dashes, and 'Process exited after 8.222 seconds with return value 0'. It ends with 'Press any key to continue . . .'.

```
C:\Users\Admin\Desktop\prai > Enter a number:143
Sum is=8
-----
Process exited after 8.222 seconds with return value 0
Press any key to continue . . .
```

2. Write a program for to perform liner search.

Program:

```
#include <stdio.h>

int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i;
        }
    }

    return -1;
}
```

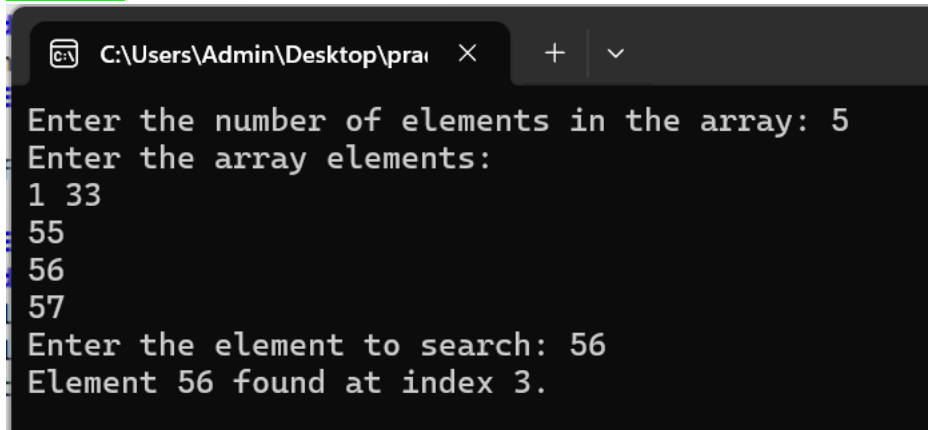
```

int main() {
    int n, key;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the array elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the element to search: ");
    scanf("%d", &key);
    int result = linearSearch(arr, n, key);
    if (result != -1)
        printf("Element %d found at index %d.\n", key, result);
    else
        printf("Element %d not found in the array.\n", key);

    return 0;
}

```

Output:



```

C:\Users\Admin\Desktop\prai >
Enter the number of elements in the array: 5
Enter the array elements:
1 33
55
56
57
Enter the element to search: 56
Element 56 found at index 3.

```

3. Write a program to perform n Queens problem using backtracking.

Program:

```

#include <stdio.h>
#include <stdbool.h>

#define MAX_N 20

```

```

void printSolution(int board[MAX_N][MAX_N], int N) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf("%2d ", board[i][j]);
        printf("\n");
    }
}

```

```

bool isSafe(int board[MAX_N][MAX_N], int row, int col, int N) {

    for (int i = 0; i < col; i++)
        if (board[row][i])
            return false;

    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;
    for (int i = row, j = col; i < N && j >= 0; i++, j--)
        if (board[i][j])
            return false;

    return true;
}

```

```

bool solveNQueensUtil(int board[MAX_N][MAX_N], int col, int N) {
    if (col >= N)
        return true;

    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col, N)) {
            board[i][col] = 1;

            if (solveNQueensUtil(board, col + 1, N))
                return true;

            board[i][col] = 0;
        }
    }
    return false;
}

```

```

    }
}

return false;
}

bool solveNQueens(int N) {
    if (N <= 0 || N > MAX_N) {
        printf("Invalid board size. Please choose a size between 1 and
%d.\n", MAX_N);
        return false;
    }

    int board[MAX_N][MAX_N] = {{0}};

    if (!solveNQueensUtil(board, 0, N)) {
        printf("Solution does not exist.\n");
        return false;
    }

    printf("Solution for N-Queens problem:\n");
    printSolution(board, N);
    return true;
}

int main() {
    int N;

    printf("Enter the size of the chessboard (N): ");
    scanf("%d", &N);

    solveNQueens(N);
    return 0;
}

```

Output:

```
Enter the size of the chessboard (N): 5
Solution for N-Queens problem:
1 0 0 0 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0
```

4. Write a program to inset a number in a list.

Program:

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

```
int main() {
    int i, size;
    printf("Enter the size of the array: ");
    scanf("%d", &size);
    if (size <= 0 || size > 1000) {
        printf("Invalid array size. Please enter a size between 1 and
1000.\n");
        return 1;
    }

    int myArray[1000];
    for (i = 0; i < size; i++) {
        myArray[i] = i;
    }
    printf("Array elements:\n");
    for (i = 0; i < size; i++) {
        printf("myArray[%d] = %d\n", i, myArray[i]);
    }

    return 0;
}
```

Output:

```
C:\Users\Admin\Desktop\prai × + v
Enter the size of the array: 5
Array elements:
myArray[0] = 0
myArray[1] = 1
myArray[2] = 2
myArray[3] = 3
myArray[4] = 4
```

5. Write a program to perform sum of subsets problem using backtracking

Program:

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

```
#include <stdbool.h>
```

```
#define MAX_SIZE 100
```

```
void printSubset(int subset[], int subsetSize) {
```

```
    printf("Subset: { ");
```

```
    for (int i = 0; i < subsetSize; i++) {
```

```
        printf("%d ", subset[i]);
```

```
    }
```

```
    printf("}\n");
```

```
}
```

```
void subsetSumUtil(int set[], int subset[], int n, int target, int sum, int
index, int subsetSize) {
```

```
    if (sum == target) {
```

```
        printSubset(subset, subsetSize);
```

```
        return;
```

```
    }
```

```
    for (int i = index; i < n; i++) {
```

```
        subset[subsetSize] = set[i];
```

```
        subsetSumUtil(set, subset, n, target, sum + set[i], i + 1, subsetSize +
1);
```

```
    }
```

```

}
void subsetSum(int set[], int n, int target) {
    int subset[MAX_SIZE];
    subsetSumUtil(set, subset, n, target, 0, 0, 0);
}

int main() {
    int n, target;
    printf("Enter the size of the set: ");
    scanf("%d", &n);

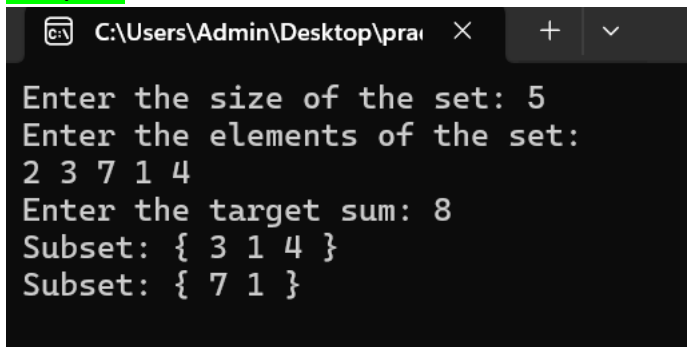
    if (n <= 0 || n > MAX_SIZE) {
        printf("Invalid set size. Please enter a size between 1 and %d.\n",
MAX_SIZE);
        return 1;
    }

    int set[MAX_SIZE];
    printf("Enter the elements of the set:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &set[i]);
    }
    printf("Enter the target sum: ");
    scanf("%d", &target);
    subsetSum(set, n, target);

    return 0;
}

```

Output:



```

C:\Users\Admin\Desktop\prai >
Enter the size of the set: 5
Enter the elements of the set:
2 3 7 1 4
Enter the target sum: 8
Subset: { 3 1 4 }
Subset: { 7 1 }

```

6. Write a program to perform graph coloring problem using backtracking.

Program:

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 10

bool isSafe(int vertex, int graph[MAX_VERTICES][MAX_VERTICES], int
colors[MAX_VERTICES], int color, int V) {
    for (int i = 0; i < V; i++) {
        if (graph[vertex][i] && color == colors[i]) {
            return false;
        }
    }
    return true;
}

bool graphColoringUtil(int graph[MAX_VERTICES][MAX_VERTICES], int
colors[MAX_VERTICES], int vertex, int V, int m) {
    if (vertex == V)
        return true;

    for (int color = 1; color <= m; color++) {
        if (isSafe(vertex, graph, colors, color, V)) {
            colors[vertex] = color;

            if (graphColoringUtil(graph, colors, vertex + 1, V, m))
                return true;

            colors[vertex] = 0;
        }
    }

    return false;
}
```



```

void graphColoring(int graph[MAX_VERTICES][MAX_VERTICES], int V, int
m) {
    int colors[MAX_VERTICES] = {0};

    if (graphColoringUtil(graph, colors, 0, V, m)) {
        printf("Graph coloring is possible with at most %d colors.\n", m);
        printf("Color assignment for each vertex:\n");
        for (int i = 0; i < V; i++) {
            printf("Vertex %d: Color %d\n", i + 1, colors[i]);
        }
    } else {
        printf("Graph coloring is not possible with %d colors.\n", m);
    }
}

int main() {
    int V, m;

    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &V);

    if (V <= 0 || V > MAX_VERTICES) {
        printf("Invalid number of vertices. Please enter a value between 1
and %d.\n", MAX_VERTICES);
        return 1;
    }

    int graph[MAX_VERTICES][MAX_VERTICES];

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

    printf("Enter the number of colors available: ");

```

```

scanf("%d", &m);

graphColoring(graph, V, m);

return 0;
}

```

Output:

```

Enter the number of vertices in the graph: 4
Enter the adjacency matrix of the graph (4 x 4):
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0
Enter the number of colors available: 3
Graph coloring is possible with at most 3 colors.
Color assignment for each vertex:
Vertex 1: Color 1
Vertex 2: Color 2
Vertex 3: Color 3
Vertex 4: Color 2

```

7. Write a program to compute container loader Problem.

Program:

```

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

typedef struct {
    int id;
    int weight;
} Item;

int compareItems(const void *a, const void *b) {
    return ((Item *)b)->weight - ((Item *)a)->weight;
}

void loadContainer(Item items[], int n, int containerCapacity) {
    qsort(items, n, sizeof(Item), compareItems);

    int currentWeight = 0;
    int containerCount = 1;

```

```

printf("Container %d: ", containerCount);

for (int i = 0; i < n; i++) {
    if (currentWeight + items[i].weight <= containerCapacity) {
        printf("%d ", items[i].id);
        currentWeight += items[i].weight;
    } else {
        printf("\nContainer %d: %d ", ++containerCount, items[i].id);
        currentWeight = items[i].weight;
    }
}

printf("\nTotal number of containers used: %d\n", containerCount);
}

int main() {
    int n, containerCapacity;

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

    Item items[n];

    for (int i = 0; i < n; i++) {
        items[i].id = i + 1;
        printf("Enter the weight of item %d: ", i + 1);
        scanf("%d", &items[i].weight);
    }

    printf("Enter the capacity of the container: ");
    scanf("%d", &containerCapacity);

    loadContainer(items, n, containerCapacity);

    return 0;
}

```

Output:

```
C:\Users\Admin\Desktop\prai X + v
Enter the number of items: 5
Enter the weight of item 1: 2
Enter the weight of item 2: 5
Enter the weight of item 3: 7
Enter the weight of item 4: 8
Enter the weight of item 5: 6
Enter the capacity of the container: 20
Container 1: 4 3
Container 2: 5 2 1
Total number of containers used: 2
```

8. Write a program to generate the list of all factor for n value using recursion

Program:

```
#include <stdio.h>

void generateFactors(int n, int i) {
    if (i > n) {
        return;
    }

    if (n % i == 0) {
        printf("%d ", i);
    }

    generateFactors(n, i + 1);
}

int main() {
    int n;

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

    if (n <= 0) {
        printf("Please enter a positive integer for n.\n");
        return 1;
    }
}
```

```

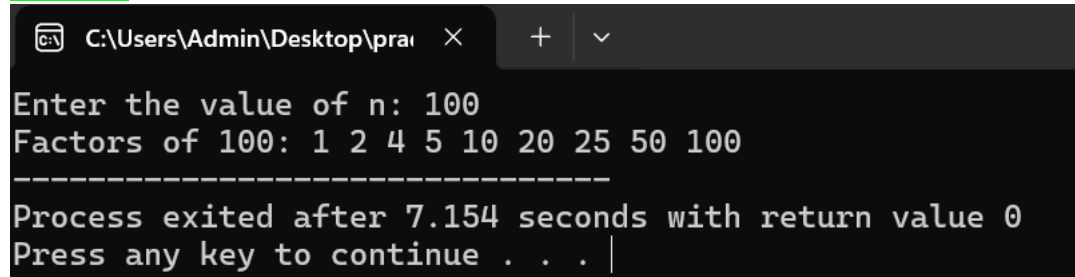
    }

    printf("Factors of %d: ", n);
    generateFactors(n, 1);

    return 0;
}

```

Output:



```

C:\Users\Admin\Desktop\prai > Enter the value of n: 100
Factors of 100: 1 2 4 5 10 20 25 50 100
-----
Process exited after 7.154 seconds with return value 0
Press any key to continue . . . |

```

9. Write a program to perform Assignment problem using branch and bound.

Program:

```

#include <stdio.h>
#include <limits.h>
#define N 10
int minCost = INT_MAX;

int isValid(int assignment[N], int worker, int job) {
    for (int i = 0; i < N; i++) {
        if (assignment[i] == job || assignment[i] == -1)
            return 0;
    }
    return 1;
}

void branchAndBound(int assignment[N], int costMatrix[N][N], int cost,
int worker) {
    if (worker == N) {

        if (cost < minCost)
            minCost = cost;
        return;
    }
}

```

```

    }

    for (int job = 0; job < N; job++) {
        if (isValid(assignment, worker, job)) {
            assignment[worker] = job;
            branchAndBound(assignment, costMatrix, cost +
costMatrix[worker][job], worker + 1);
            assignment[worker] = -1;
        }
    }
}

int main() {
    int costMatrix[N][N];
    int assignment[N];

    printf("Enter the cost matrix (%d x %d):\n", N, N);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            scanf("%d", &costMatrix[i][j]);
        }
    }

    for (int i = 0; i < N; i++) {
        assignment[i] = -1;
    }
    branchAndBound(assignment, costMatrix, 0, 0);
    printf("Minimum cost of assignment: %d\n", minCost);

    return 0;
}

```

10. Write a program to find out Hamiltonian circuit Using backtracking method

Program:

```
#include <stdio.h>

#define V 5

void printSolution(int path[V]);

int isSafe(int v, int pos, int path[V], int graph[V][V]) {
    if (graph[path[pos - 1]][v] == 0)
        return 0;

    for (int i = 0; i < pos; i++)
        if (path[i] == v)
            return 0;

    return 1;
}

int hamiltonianUtil(int graph[V][V], int path[V], int pos) {
    if (pos == V) {
        if (graph[path[pos - 1]][path[0]] == 1)
            return 1;
        else
            return 0;
    }

    for (int v = 1; v < V; v++) {
        if (isSafe(v, pos, path, graph)) {
```

```

        path[pos] = v;

        if (hamiltonianUtil(graph, path, pos + 1) == 1)
            return 1;

        path[pos] = -1;
    }
}

return 0;
}

int hamiltonianCycle(int graph[V][V]) {
    int path[V];
    for (int i = 0; i < V; i++)
        path[i] = -1;

    path[0] = 0;
    if (hamiltonianUtil(graph, path, 1) == 0) {
        printf("Solution does not exist.\n");
        return 0;
    }

    printSolution(path);
    return 1;
}

void printSolution(int path[V]) {

```



```

printf("Hamiltonian Cycle: ");
for (int i = 0; i < V; i++)
    printf("%d ", path[i]);

printf("%d", path[0]);
printf("\n");
}

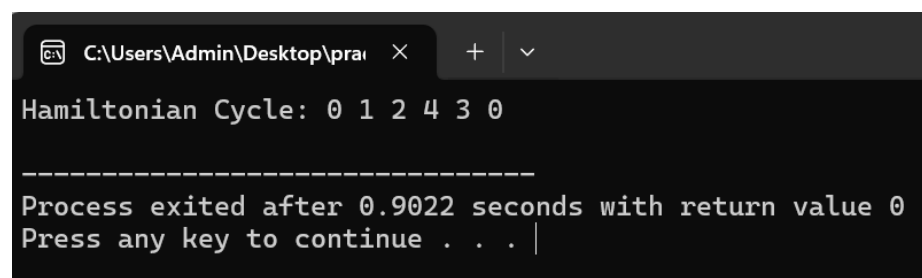
int main() {
    int graph[V][V] = {
        {0, 1, 0, 1, 0},
        {1, 0, 1, 1, 1},
        {0, 1, 0, 0, 1},
        {1, 1, 0, 0, 1},
        {0, 1, 1, 1, 0}
    };

    hamiltonianCycle(graph);

    return 0;
}

```

Output:



```

C:\Users\Admin\Desktop\prai × + ▾
Hamiltonian Cycle: 0 1 2 4 3 0
-----
Process exited after 0.9022 seconds with return value 0
Press any key to continue . . . |

```