Experiment-9

Program:

```
#include <stdio.h>

void findSubset(int S[], int n, int d, int subset[], int
size, int sum);

int main()
{
    int S[] = {1, 2, 3};
    int n = sizeof(S) / sizeof(S[0]);
    int d = 3;
    int subset[n];
    int size = 0;
    int sum = 0;
    printf("Given items in the SET: ");
    for(int i=0;i<n;i++)
        printf("%d ",S[i]);
    printf("\nDesirable 'SUM' : %d",d);
    printf("\nFollowing are the Subsets found: \n");
    findSubset(S, n, d, subset, size, sum);
    if (size == 0)
    {
        printf("No more subsets found.\n");
    }
    return 0;
}

void findSubset(int S[], int n, int d, int subset[], int
size, int sum)
{
    if (sum == d)
    {
        printf("{");
        for (int i = 0; i<size; i++)
        {
            printf("%d", subset[i]);
            if (i < size-1)
            {
                printf(",");
            }
        }
        printf("}\n");
        return;
    }
    if (sum > d || n == 0)
    {
        return;
    }
    subset[size] =  S[0];
```

```
                findSubset(S+1, n-1, d, subset, size+1, sum+S[0]);
                findSubset(S+1, n-1, d, subset, size, sum);
            }
```

Output:

```
Given items in the SET: 1 2 5 6 8
Desirable 'SUM' : 9
Following are the Subsets found:
{1,2,6}
{1,8}
No more subsets found.
```

Experiment-10

Program:

```
#define N 4
#include <stdbool.h>
#include <stdio.h>

// A utility function to print solution
void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if(board[i][j])
                printf("Q ");
            else
                printf(". ");
        }
        printf("\n");
    }
}

// A utility function to check if a queen can be placed on
board[row][col].
bool isSafe(int board[N][N], int row, int col)
{
    int i, j;

    // Check this row on left side
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;
    // Check upper diagonal on left side
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;

    // Check lower diagonal on left side
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}

// A recursive utility function to solve N
// Queen problem
bool solveNQUtil(int board[N][N], int col)
{
    // Base case: If all queens are placed then return true
    if (col >= N)
        return true;

    // Consider this column and try placing this queen in all rows
one by one
    for (int i = 0; i < N; i++) {

        // Check if the queen can be placed on board[i][col]
        if (isSafe(board, i, col)) {

            // Place this queen in board[i][col]
            board[i][col] = 1;

            // Recur to place rest of the queens
            if (solveNQUtil(board, col + 1))
                return true;

            board[i][col] = 0; // BACKTRACK
        }
    }

    // If the queen cannot be placed in any row in
    // this column col  then return false
    return false;
}

bool solveNQ()
{
    int board[N][N] = { { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 } };
    if (solveNQUtil(board, 0) == false)
    {
        printf("Solution does not exist");
        return false;
    }

    printSolution(board);
    return true;
}

// Driver program to test above function
int main()
{
    printf("Solution for 4-Queens Problem:\n\n");
    solveNQ();
    return 0;
}
```

Output:

```
Solution for 4-Queens Problem:

. . Q .
Q . . .
. . . Q
. Q . .
```