Name	Mohammed Muzammil Ansari
UID no.	2022701001
Experiment No.	5

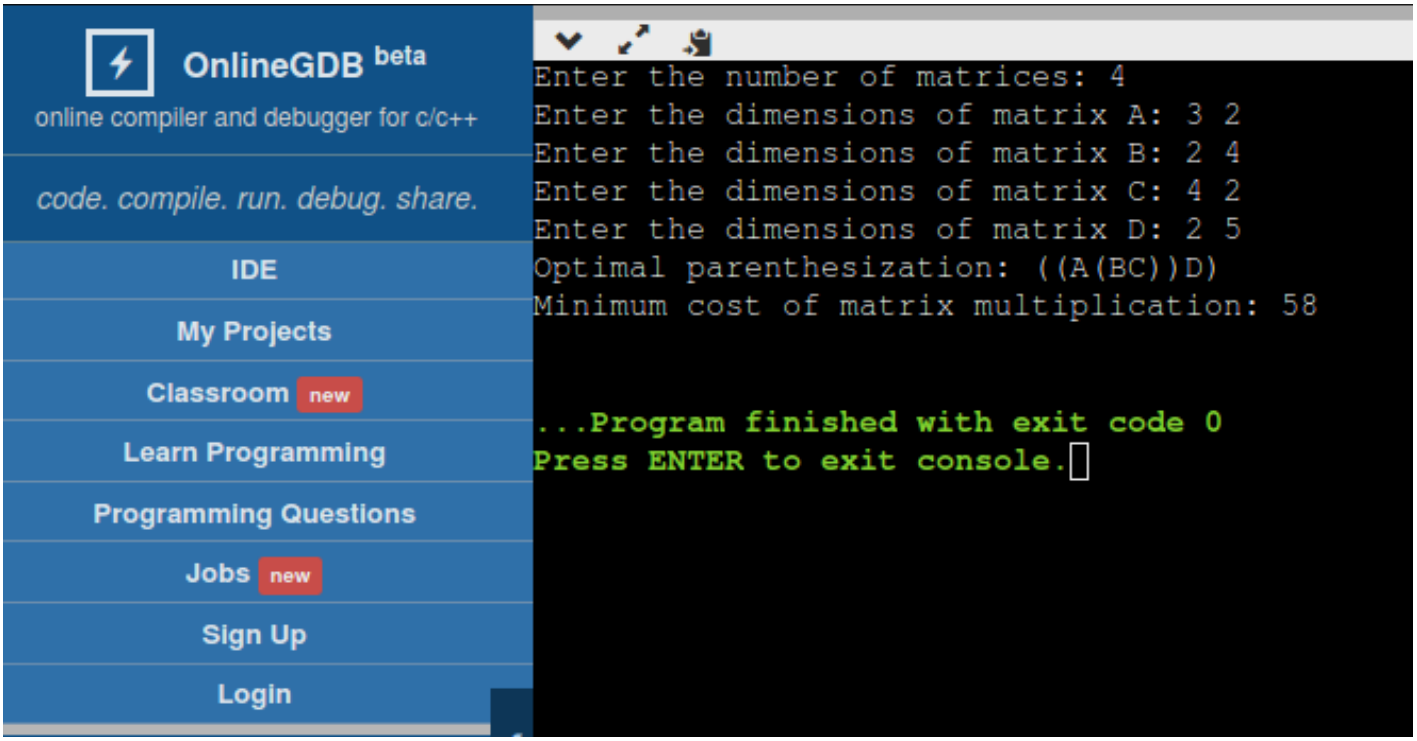
AIM:	Experiment on implementing matrix chain multiplication algorithm.
Program 1	
Algorithm:	1) Define a function, "matrix_chain_order", that takes in a list of matrix dimensions, "p", where the i-th matrix has dimensions p[i-1] x p[i]. 2) Initialize a two-dimensional array, "m", of size n x n, where n is the length of p - 1 (i.e. the number of matrices in the chain). 3) Initialize a two-dimensional array, "s", of size n x n, where s[i][j] will eventually store the index of the matrix that should be used as the split point between matrices i and j in the optimal solution. 4) For each diagonal element in m, set m[i][i] = 0. 5) For each sub-chain length (l) from 2 to n, do the following: a. For each i from 1 to n - l + 1, do the following: i. Set j = i + l - 1. ii. Set m[i][j] to infinity. iii. For each k from i to j - 1, do the following: 1. Set q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j]. 2. If q is less than m[i][j], set m[i][j] = q and s[i][j] = k. 6) Return m and s.

PROGRAM:

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

#define MAX_SIZE 100

// function to print the optimal parenthesization of a matrix chain
void print_optimal_parens(int s[MAX_SIZE][MAX_SIZE], int i, int j, char name) {
        if (i == j) {
        printf("%c", name++);
        } else {
        printf("(");
        print_optimal_parens(s, i, s[i][j], name);
        print_optimal_parens(s, s[i][j]+1, j, name+s[i][j]-i+1);
        printf(")");
        }
}

// function to compute the minimum cost of matrix multiplication using dynamic
programming m and s  are cost and k table.
int matrix_chain_order(int p[], int n, char name) {
        int m[MAX_SIZE][MAX_SIZE], s[MAX_SIZE][MAX_SIZE];

        for (int i = 1; i <= n; i++) {
        m[i][i] = 0;
        }

        for (int l = 2; l <= n; l++) {
        for (int i = 1; i <= n - l + 1; i++) {
        int j = i + l - 1;
        m[i][j] = INT_MAX;
        for (int k = i; k <= j - 1; k++) {
        int q = m[i][k] + m[k+1][j] + p[i-1] * p[k] * p[j];
        if (q < m[i][j]) {
                m[i][j] = q;
                s[i][j] = k;
        }
        }
        }
        }
```

```
        printf("Optimal parenthesization: ");
        print_optimal_parens(s, 1, n, name);
        printf("\n");

        return m[1][n];
}

int main() {
        int num_matrices;
        printf("Enter the number of matrices: ");
        scanf("%d", &num_matrices);

        int matrices[num_matrices][2]; // assuming each matrix has 2 dimensions

        // loop through each matrix and get its dimensions
        for (int i = 0; i < num_matrices; i++) {
        printf("Enter the dimensions of matrix %c: ", 'A' + i);
        scanf("%d %d", &matrices[i][0], &matrices[i][1]);
        }

        // create a 1D array of matrix dimensions
        int matrix_sizes[MAX_SIZE];
        int idx = 0;
        for (int i = 0; i < num_matrices; i++) {
        matrix_sizes[idx++] = matrices[i][0];
        if (i == num_matrices - 1) {
        matrix_sizes[idx++] = matrices[i][1];
        }
        }

        // compute the minimum cost and optimal parenthesization using dynamic
programming
        printf("Minimum cost of matrix multiplication: %d\n",
matrix_chain_order(matrix_sizes, idx - 1, 'A'));

        return 0;



}
```

RESULT:



CONCLUSION:	Thus, we have implemented a matrix chain multiplication algorithm to get the minimum cost of multiplication and optimal parenthesization.