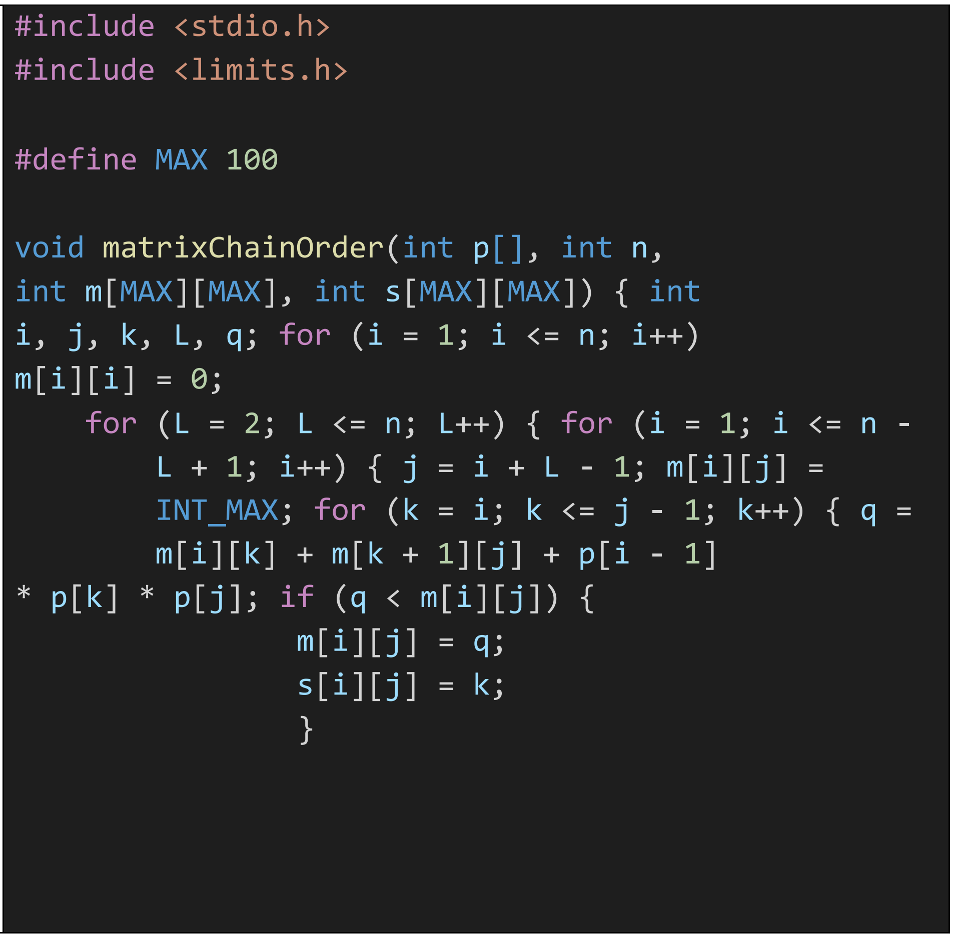
Name	**Gitanjali Gangurde**
UID	**2021300034**
Class	**SE-Comps A Batch-B**
Experiment No.	**4**
AIM:	To find the minimum matrix chain multiplications required.
ALGORITHM:	MATRIX-CHAIN-ORDER (p) 1. n length[p]-1 2. for i ← 1 to n 3. do m [i, i] ← 0 4. for l ← 2 to n // l is the chain length 5. do for i ← 1 to n-l + 1 6. do j ← i+ l -1 7. m[i,j] ← ∞ 8. for k ← i to j-1 9. do q ← m [i, k] + m [k + 1, j] + pi-1 pk pj 10. If q < m [i,j] 11. then m [i,j] ← q 12. s [i,j] ← k 13. return m and s

CODE:



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

#define MAX 100

void matrixChainOrder(int p[], int n,
int m[MAX][MAX], int s[MAX][MAX]) { int
i, j, k, L, q; for (i = 1; i <= n; i++)
m[i][i] = 0;
    for (L = 2; L <= n; L++) { for (i = 1; i <= n -
        L + 1; i++) { j = i + L - 1; m[i][j] =
        INT_MAX; for (k = i; k <= j - 1; k++) { 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;
                }
```

```
            }
        }
    }
}

void printOptimalParentheses(int s[MAX][MAX], int
i, int j) { if (i == j) { printf("A%d ", i);
    } else { printf("(");
        printOptimalParentheses(s, i, s[i][j]);
        printOptimalParentheses(s, s[i][j] + 1,
        j); printf(")");
    }
}

int main() {
    int n, i,
    j;
    int p[MAX], m[MAX][MAX], s[MAX][MAX];

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

    printf("Enter the dimensions of the
matrices:\n"); for (i = 0; i
    <= n; i++) { scanf("%d",
    &p[i]);
    }

    matrixChainOrder(p, n, m, s);
     printf("\n P= (");
    for (i = 0; i <= n; i++) {
        printf("%d ", p[i]);

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


    printf("\nM Table:");
    printf("\n");
    for(int i=n;i>=1;i--)
    {
          for(int j=1;j<=n;j++)
          { printf("%d\t",m[i][j]);
          }
          printf("\n");

      }

      printf("\nS Table:");
      printf("\n");
      for(int i=1;i<=n;i++)
      {

          for(int j=1;j<=n;j++)
          { printf("%d\t",s[i][j]);
          }
          printf("\n");

      }

      printf("\nOptimal Parenthesization is: ");
      printOptimalParentheses(s, 1, n);

      printf("\nMinimum number of scalar
multiplications: %d", m[1][n]);

      return 0;
}
```

```
}
```

OUTPUT:

```
Enter the number of matrices: 6
Enter the dimensions of the 5   ices:
10 3 12 5 50 6


 P= (5 10 3 12 5 50 6 )


M Table:
0        0       0       0       0       0
0       0       0       0       0       1500

0       0       0       0       3000    1860

0       0       0       180     930     1770
```

```
0       0       360     330     2430    1950

0       150     330     405     1655    2010


S
Table:
0       1       2       2       4       2
0       0       2       2       2       2

0       0       0       3       4       4

0       0       0       0       4       4

0       0       0       0       0       5

0       0       0       0       0       0


Optimal Parenthesization is: ((A1 A2 )((A3 A4 )(A5 A6 )))
Minimum number of scalar multiplications: 2010
```

CONCLUSION:

1. This dynamic programming approach reduces time complexity of naïve method of matrix chain multiplication.
2. The time complexity of matrix chain multiplication is O(n^3).
3. The space complexity of matrix chain multiplication is O(n^2).