MatrixChain.cpp

```cpp
// See the Cormen book for details of the following algorithm
#include<stdio.h>
#include<limits.h>

int MatrixChainOrderR(int p[], int i, int j) {
    if(i == j)
        return 0;
    int k;
    int min = INT_MAX;
    int count;

    // place parenthesis at different places between first
    // and last matrix, recursively calculate count of
    // multiplications for each parenthesis placement and
    // return the minimum count
    for (k = i; k <j; k++) {
        count = MatrixChainOrderR(p, i, k) +
                MatrixChainOrderR(p, k+1, j) +
                p[i-1]*p[k]*p[j];

        if (count < min) min = count;
    }

    // Return minimum count
    return min;
}

// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
int MatrixChainOrder(int p[], int n) {
    /* For simplicity of the program, one extra row and one
       extra column are allocated in m[][].  0th row and 0th
       column of m[][] are not used */
    int m[n][n];

    int i, j, k, L, q;

    /* m[i,j] = Minimum number of scalar multiplications needed
       to compute the matrix A[i]A[i+1]...A[j] = A[i..j] where
       dimension of A[i] is p[i-1] x p[i] */

    // cost is zero when multiplying one matrix.
    for (i=1; i<n; i++) m[i][i] = 0;

    // L is chain length.
    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 = cost/scalar multiplications
                q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
                if (q < m[i][j])
                    m[i][j] = q;
            }
        }
    }

    return m[1][n-1];
}

int main()
```

```
{
    int arr[] = {1, 2, 3, 4};
    int size = sizeof(arr)/sizeof(arr[0]);

    printf("Minimum number of multiplications is %d ",
                      MatrixChainOrder(arr, size));

    getchar();
    return 0;
}
```