

Exercise 3-6

Maximilian Fernaldy - C2TB1702

Exercise 3-6: Matrix calculation for multidimensional arrays matrix_multiply.c 15

- Create a program `matrix_multiply.c` that finds the product C of the 4-by-3 matrix A and the 3-by-4 matrix B .
- $C = A \times B$

$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$

$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \end{pmatrix}$

The matrix values have already been entered in the sample source `matrix_multiply.c`, so please use it.

points:

1. Be aware of which is the row and which is the column when coding
`int A[4][3]`
row column
2. Matrix subscripts start at 0

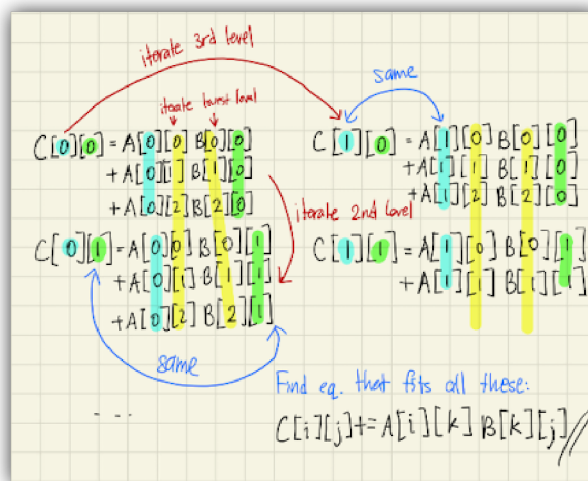
<example>

```
$ ./matrix_multiply
C[0][0] = 14
C[0][1] = 20
C[0][2] = 26
C[0][3] = 32
C[1][0] = 35
C[1][1] = 50
C[1][2] = 65
C[1][3] = 80
C[2][0] = 14
C[2][1] = 20
C[2][2] = 26
C[2][3] = 32
C[3][0] = 35
C[3][1] = 50
C[3][2] = 65
C[3][3] = 80
```

```
$ ./matrix_multiply
C =
| 14 20 26 32|
| 35 50 65 80|
| 14 20 26 32|
| 35 50 65 80|
```

Even though it might seem complicated at first, matrix multiplication is fairly simple with `for` loops, especially when we know the dimensions of the matrices and we don't need to implement adaptivity.

Since we know that the program will encounter a 4×3 matrix A and 3×4 matrix B , it is mathematically deducible that the resulting matrix C will be a 4×4 matrix. To derive the mathematical representation of the multiplication, I wrote down the first few iterations of the multiplication, found the pattern, and translated it into code.



From the different additive multiplications, we can see a pattern form. For any given entry c_{ij} , it is equal to the sum of the product of a_{ik} and b_{kj} . We can represent this mathematically as:

$$c_{ij} = \sum_{k=1}^3 a_{ik} \times b_{kj}$$

Note that this formula is obviously only valid for this case. A more general formula would be

$$c_{ij} = \sum_{k=1}^n a_{ik} \times b_{kj}$$

for the multiplication of matrices with size $m \times n$ and $n \times p$.

If we translate this into code, the sum of $a_{ik} \times b_{kj}$ can be obtained by iterating k from 0 to 2. We can start with 1 to 3 just as well, but I decided to stay with the general formula used to define `for` loops in C, which is to start with 0 for the iterator variable. This is the lowest level of the iteration, used to define a single element c_{ij} . Next, we need to fill up the matrix, so we need two more levels of iteration, one to go right and fill a single row, then go down and fill the whole matrix. We use the j iterator and the i iterator respectively to do these things. After these nested `for` loops we should end up with all the entries for matrix C .

```

for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
        for (int k = 0; k < 3; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}

```

To print the output we must use more nested `for` loops, this time only 2 levels deep because we just need 2 iterators.

```

/* Display results */
// Dropdown
printf("Dropdown view:\n");
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
        printf("C[%d][%d] = %d\n", i, j, C[i][j]);
    }
}

```

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     int A[4][3] = {{1,2,3},{4,5,6},{1,2,3},{4,5,6}};
6     int B[3][4] = {{1,2,3,4},{5,6,7,8},{1,2,3,4}};
7     int C[4][4] = {0};
8
9     for (int i = 0; i < 4; i++) {
10        for (int j = 0; j < 4; j++) {
11            for (int k = 0; k < 3; k++) {
12                C[i][j] += A[i][k] * B[k][j];
13            }
14        }
15    }
16
17    for (int i = 0; i < 4; i++) {
18        for (int j = 0; j < 4; j++) {
19            printf("C[%d][%d] = %d\n", i, j, C[i][j]);
20        }
21    }
22 }

```

```

max ex3-6 % ./nestedFor
C[0][0] = 14
C[0][1] = 20
C[0][2] = 26
C[0][3] = 32
C[1][0] = 35
C[1][1] = 50
C[1][2] = 65
C[1][3] = 80
C[2][0] = 14
C[2][1] = 20
C[2][2] = 26
C[2][3] = 32
C[3][0] = 35
C[3][1] = 50
C[3][2] = 65
C[3][3] = 80
max ex3-6 %

```

To print the output in a more easily read, graphical style we can do

```

// Visual
printf("Visual view:\n");
for (int i = 0; i < 4; i++) {
    printf("| "); // Left boundary
    for (int j = 0; j < 4; j++) {
        printf(" %d ", C[i][j]); // print entry
    }
    printf(" |\n"); // Right boundary, then insert newline for new row
}

```

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     int A[4][3] = {{1,2,3},{4,5,6},{1,2,3},{4,5,6}};
6     int B[3][4] = {{1,2,3,4},{5,6,7,8},{1,2,3,4}};
7     int C[4][4] = {0};
8
9     for (int i = 0; i < 4; i++) {
10        for (int j = 0; j < 4; j++) {
11            for (int k = 0; k < 3; k++) {
12                C[i][j] += A[i][k] * B[k][j];
13            }
14        }
15    }
16
17    for (int i = 0; i < 4; i++) {
18        printf("| ");
19        for (int j = 0; j < 4; j++) {
20            printf(" %d ", C[i][j]);
21        }
22        printf(" |\n");
23    }
24 }

```

```

max ex3-6 % ./matrix_multiple
Dropdown view:
C[0][0] = 14
C[0][1] = 20
C[0][2] = 26
C[0][3] = 32
C[1][0] = 35
C[1][1] = 50
C[1][2] = 65
C[1][3] = 80
C[2][0] = 14
C[2][1] = 20
C[2][2] = 26
C[2][3] = 32
C[3][0] = 35
C[3][1] = 50
C[3][2] = 65
C[3][3] = 80
Visual view:
| 14 20 26 32 |
| 35 50 65 80 |
| 14 20 26 32 |
| 35 50 65 80 |
max ex3-6 %

```