

Introduction to Programming in C
Prof. Satyadev Nandakumar
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 14

In this session, let us look at a matrix problem and the problem is given as follows.

(Refer Slide Time: 00:06)

A Matrix Problem

- The first line of the input consists of two positive integers m and n .
- This line is followed by m lines, each containing n integers, signifying an $m \times n$ matrix A . We have to calculate the sum of the squares of the sum of numbers in each row and print it.

$$\sum_i (\sum_j A_{ij})^2, i=0..m-1, j=0..n-1.$$

3 4
4 7 11 2
1 1 2 4
2 9 0 -1

row i ↓
columns j →

e.g. $A_{20} = 2, A_{12} = 2$

Desired output

$$(4+7+11+2)^2 + (1+1+2+4)^2 + (2+9+0+(-1))^2$$

You have a bunch of lines of input. The first line of the input consists of two numbers, m and n . This line is followed by m lines, each containing n integers. So, this is supposed to represent the matrix of size m time m cross n . We have to calculate the sum of the squares of the sum of numbers in each row, that is quite a mouthful, let us look at the formula. So, what I want to do is, summation $j A_{ij}$. So, once you do this sum, you square that and then do the summation i .

So, i is an index which goes over the rows. In C, we adopt the convention that the first row is starting with 0 and so, it goes from 0 to n minus 1. Similarly, the first column starts with index 0, so it goes on till n minus 1. So, the input will be given as follows. The first number m represents that there are 3 rows, the second number represents the number of columns in each row and then this is followed by a matrix of size 3 cross 4.

Now, the numbering is given as follows. For example, if you have A_{20} , this means second row zeroth column. Note that, second row means we start with row 0, then row 1,

then row 2 and zeroth column is this, the first column. So, A_{20} is this number which is 2. Similarly, A_{12} is first row, row number 1 column number 2, which is also 2. So, the desired output that we have is 4 plus 7 plus 11 plus 2 whole square and so on. So, how do we do this by hand? Let us just look at the calculation.

(Refer Slide Time: 02:15)

$$\sum_{i=0}^2 \left(\sum_{j=0}^3 A_{ij} \right)^2$$

4	7	11	2	$21^2 = 576$
1	1	2	4	64
2	9	0	-1	<u>100</u>

$$\sum_{i=0}^2 \left(\sum_{j=0}^3 A_{ij} \right)^2 = 740$$

So, we have 4 7 11 2 1 1 2 4 and 2 9 0 minus 1. Notice that the formula that we have to calculate is i going from 0 to 2, j going from 0 to 3 A_{ij} squared. So, how do we do this? First, we sum the numbers in each column so, I will name that as j going from 0 to 3 A_{ij} and then square. So, if you sum all this, you see 11 22 24 square which is 576 and similarly, this is 64 8 square and this is 10 square, which is 100.


So, what I have tallied on the right hand side is, for each row you sum the numbers take that sum and square it. And then finally in order to compute what we want, which is i equal to 0 to 2 of summation j equal to 0 to 3 A_{ij} square. In order to calculate this, all we need to do is to sum these numbers up and this turns out to be 740.

So, notice when we did by hand, we did the following, we first calculated row wise, we summed over all the numbers in that row. Take the sums, square it. Then, you repeated the same operation for the next row and then for the third row. So, we have three numbers and then we added them in sequence. So, we will see how we will code this up?

(Refer Slide Time: 04:22)

Double loops

- Need something of a double loop here (loop inside a loop).
- One loop to do the row sum of each row.
- Once a row is finished, we square the row sum.
- Another (outer) loop to add the squares of row sum over all rows that have been fully read.



So, what we need here is something called an inner loop or a double loop, we need a loop inside a loop. Now, the inner loop is doing what we did first? It is taking a row and adding all the numbers in that row, then squaring it. So, we need one loop to do the row sum of each row. Once a row is finished, we square the row sum. Once that is done, remember that once we tallied numbers on the right hand side and squared them. We have to add those numbers up. So, we need another loop, an outer loop to add these squares of rows sums.

(Refer Slide Time: 05:07)

Inner loop: Row sum

- Easy part first: assume we are at the beginning of a row (have not read any numbers yet) and write a loop to calculate the row sum.

```
int a;           /* the current integer */
int colindex;   /* index of current column */
int rowsum=0;   /* sum of row entries read so far */
int rowsumsq=0; /* square of the sum of row entries */

while (colindex < n) { /* not finished reading n cols*/
    scanf("%d", &a);   /* read next number */
    rowsum = rowsum + a; /* add to rowsum */
    colindex = colindex + 1; /* increment colindex */
}

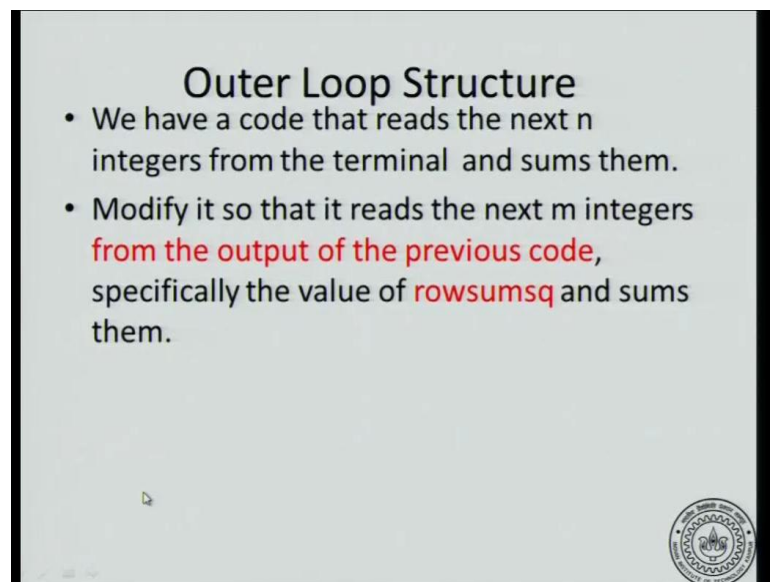
rowsumsq = rowsum * rowsum; /*square rowsum */
```

So, let us do this program in stages. First, let us write the inner loop, this is the loop, so that for a given row you sum up all the numbers in that row. Now, let us assume that we are at the beginning of a row and we have not read any numbers yet. Now, what we have to do is to start reading the numbers. So, we write a while loop. We declare four variables, a, column index, then we need something for the row sum and some integer variable for row sum square.

Now, what you do is you go along the row and add the numbers in each column. So, while the column index is less than n, recall that the matrix size was m cross n. You scan the next number, the next number is added to the row sum and then increment the column index. Until you hit n, recall that the last column is n minus 1, because we start the column numbers from 0.

Now, once you have done you have the sum of the numbers in that row and what you need to do is to square that number. So, we have row sum times row sum will be row sum squared.

(Refer Slide Time: 06:33)



That completes the inner loop, which is what we did, when we added the numbers along a given row and then finally, squared the sum. Now, what we need is an outer loop structure over these. So, we have a code that reads the next n integers from the terminal and sums them. Now, what we need is some further code, that takes the output of the previous code and then sums all those numbers up.

Remember, when we did this by hand, this was the second operation we did, we went over the right most column and added all those numbers up and that was the result that we wanted.

(Refer Slide Time: 07:13)

- Task: Modify code below so that it reads the next m integers **from the output of the previous code**, specifically the value of **rowsumsq** and sums them.

```
int a;          /* the current integer */
int colindex;  /* index of current column */
int rowsum = 0; /* sum of row entries read so far */
int rowsumsq = 0; /* square of the sum of row entries */

while (colindex < n) { /* not finished reading n cols*/
    scanf("%d", &a); /* read next number */
    rowsum = rowsum + a; /* add to rowsum */
    colindex = colindex + 1; /* increment colindex */
}
rowsumsq = rowsum * rowsum; /*square rowsum */
```

So, how do we modify the code?

(Refer Slide Time: 07:17)

- Previous code modified to read the next m integers **from the output of the previous code**, specifically the value of **rowsumsq** and

Outer Loop: Still in Design Phase: incomplete and informal

```
int rowindex=0; /* index of current row being read */
int sqsum=0; /* sum of col entries read so far */

while (rowindex < m) { /* not finished reading m rows*/
    sqsum=sqsum+`rowsumsq`^2; /* add to colsum */
    rowindex = rowindex + 1; /* increment colindex */
}
printf("%d ",colsum);
```

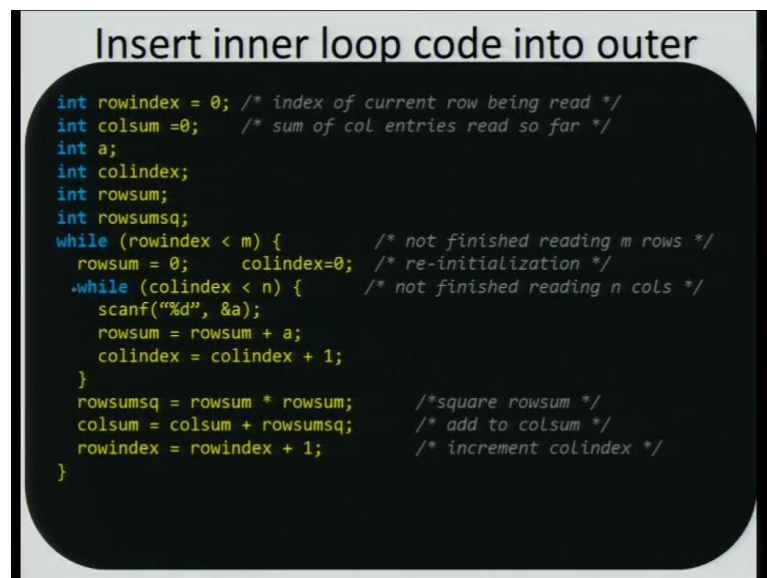
rowsumsq comes from previous code. Let's insert that code here.

So, let us what we need is something like this. Assume that we have the output available from the previous code in some variable called rowsumsquared. And we need a loop over

that, which is going from row index 0 to m minus 1 and tallying up all the numbers in rowsumsquared. So, for each row you will end up with a rowsumsquare and you have to add all those rowsumsquared to get in. So, in this the rowsumsquare comes from the previous code.

So, this is how we will visualize the outer loop. Now, note that this is not completely specified code. This is just a very intuitive picture, that instead of row sum square, it should come from some inner loop which actually calculates it. So, we should plug in the output from the previous inner loop and this is the outer loop over it.

(Refer Slide Time: 08:36)



```
int rowindex = 0; /* index of current row being read */
int colsum =0; /* sum of col entries read so far */
int a;
int colindex;
int rowsum;
int rowsumsq;
while (rowindex < m) { /* not finished reading m rows */
    rowsum = 0; colindex=0; /* re-initialization */
    while (colindex < n) { /* not finished reading n cols */
        scanf("%d", &a);
        rowsum = rowsum + a;
        colindex = colindex + 1;
    }
    rowsumsq = rowsum * rowsum; /*square rowsum */
    colsum = colsum + rowsumsq; /* add to colsum */
    rowindex = rowindex + 1; /* increment colindex */
}
```

So, here is how we put these two loops together, we have a while loop inside the while loop. Remember that, this was the previous loop that we had written. So, this was the inner loop that we have written, where at the end of the inner loop you have the rowsumsquare. Now, at the end of the code what you will end up is the exact rowsumsquare of a particular column. So, you can imagine that after this inner loop finishes execution, the rowsumsquare is the correct square of the sum of the elements in the row.

Now, the outer index does the following, you start from row index 0 and sum the rowsumsquare over all columns. So, this is the structure of the code. Let us look at the code in slightly more detail. We have a row index, a column index, a column sum, a row sum and the row sum square, a is supposed to be the current number that we are read in.

Suppose, we know that the size of the matrix is m cross n. So, row index can go from 0 to m minus 1. So, the termination condition of this while loop is row index equal to m. Now, for all that you are currently at a particular row. So, you have to initialize the row sum to 0 and the column index to 0. Because, for a given row you have to start from column 0 and you go on, until column n minus 1.

Also this previous row sum should not influence the next row. So, for every row you have to initialize the row sum variable. Once that is done, you go over the columns of that given row and you scan the numbers add the number to the row sum and increment the column index, until you hit n columns, column 0 through n minus 1. When you read all the columns in that row, you have the correct row sum. So, that row sum square can now be calculated.

Once rowsumsquare is calculated, you have the row sum square for that particular row. So, add the rowsumsquared to the previously computed columns. So, If you go back and think about how you did this by hand, you can convince yourself that this is exactly the coding of the logic that we had earlier.

(Refer Slide Time: 11:25)

The slide displays the following C code on the left:

```

int m,n;
int rowindex = 0;
int sqsum =0;
int a, colindex, rowsum;
scanf("%d,%d", &m,&n);
while (rowindex < m) {
    rowsum = 0;
    colindex=0;
    while (colindex < n) {
        scanf("%d", &a);
        rowsum=rowsum+a;
        colindex=colindex+1;
    }
    sqsum=sqsum +
        (rowsum * rowsum);
    rowindex=rowindex+1;
}

```

On the right, the input is shown as "Input 2 3" followed by a 2x3 matrix:

1	0	-1
0	1	1

The output is "Output should be 4". Below the matrix, variables are tracked:

a	rowindex	colindex	rowsum	sqsum
1	0	0	0	0
0	1	1	1	0
-1	2	2	1	4
0		3	0	
1		0	0	
1		1	0	
	3	2	1	2

So, let us try this on a small example to see exactly, how the code works? Suppose, the input is 2 3 followed by two rows of three numbers each. So, the input matrix size is 2 cross 3 and the entries are 1 0 minus 1 and 0 1 1. Let us see, how the code executes on this? So, the output should be 4, if you do it by hand and let us see, the variables are m,

n, a, row index, column index, row sum and square sum. Finally, the result should be in square sum.

You start with row index equal to 0, column index equal to 0 and you scan m and n. So, you already know the size of the matrix, when you scanned m and n. So, m becomes 2 and n becomes 3. Now, row index is 0 which is less than 2. So, it starts the loop which reads the row 0. So, notice the arrow here, you are starting to read this particular row, the first row, which is row 0. Or you initialize row sum equal to 0, column index equal to 0 and while column index is less than n, you scan the next number which is 1.

Add a to the row sum. So, row sum becomes 1, increment the column index. So, it reach column 1 row 0 read that number. Add it to the row sum, go to the second column and read the number and add in to the row sum. So, once you are done, now column index is 3. So, just means that we have read all the entries in the row 0. So, we have got the correct row sum. What we will do is, add the row sum square to the square sum.

So, row sum is 0, 0 square to square sum, so square sum remains 0. Now, you go to the second row. So, increment row index. Now, row index is less than 2 row index is 1. So, it is less than 2 we are reading row 1 and you repeat the same execution. We reinitialize the row sum to 0, column index to 0. and then, scan the next number which is 0. Add it to the row sum, increment the column index, scan the next number which is 1 and so on, until you finish reading the second row as well.

So, once you read the second row, you will find that the row sum is 2 and square sum would be square sum plus 2 square which is 4. After you do that, you increment row index and row index becomes 3, which is greater than the given row index. So, you exit the loop. So, we have correctly computed the sum that we wanted.