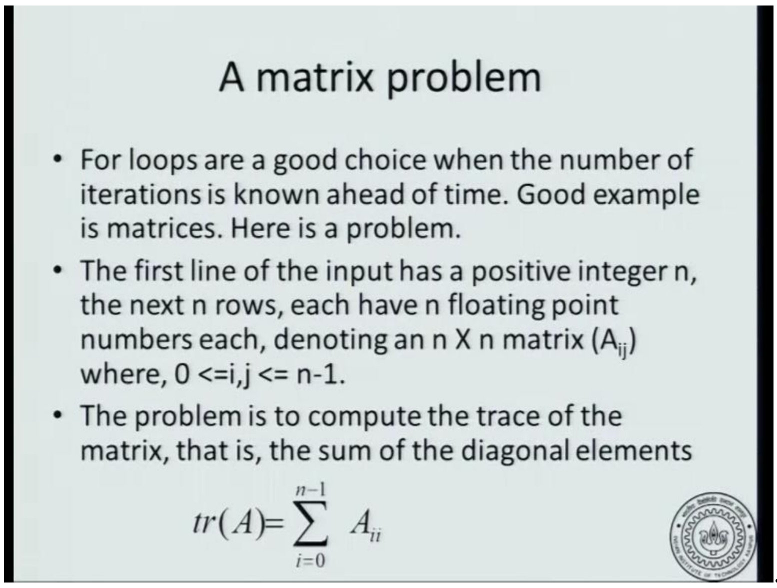
Introduction to Programming in C

Prof. Satyadev Nandakumar
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 16

In this section, we will use the, for loop to code of the matrix problem. So, remember that we have seen while loop and we have seen a do while loop. Inside while loops we have written nested loops or double loops. So, let us look at a for loop which problem, where the solution involves a nested loop.

(Refer Slide Time: 00:25)



A matrix problem

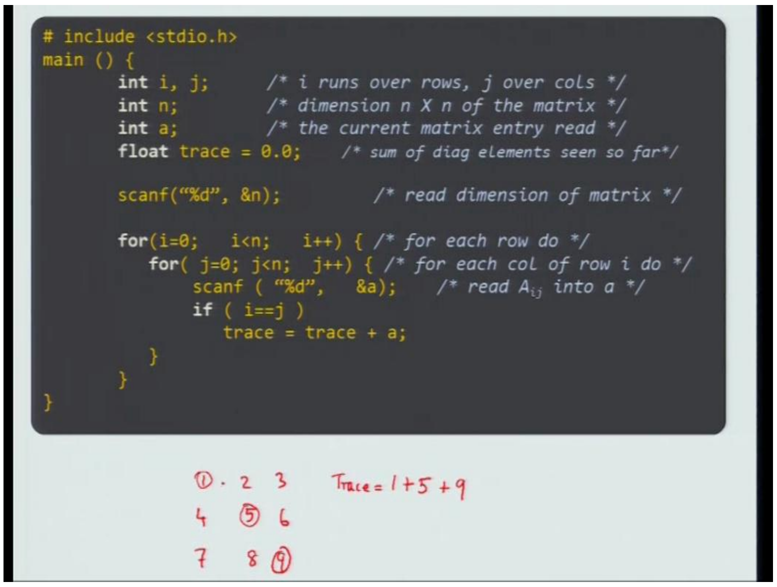
- For loops are a good choice when the number of iterations is known ahead of time. Good example is matrices. Here is a problem.
- The first line of the input has a positive integer n, the next n rows, each have n floating point numbers each, denoting an n X n matrix (A_{ij}) where, 0 <=i,j <= n-1.
- The problem is to compute the trace of the matrix, that is, the sum of the diagonal elements

$$tr(A)=\sum_{i=0}^{n-1} A_{ii}$$

.

So, the for loops are a good choice when the number of iterations is known in advance. So, a good example of such a condition is when you program for matrices, because the dimensions of a matrices are known in advance. So, let us consider a sample program. So, the first line of the input has a number n now the matrix size is n cross n and there are n floating point numbers in the matrix given row by row, each line contains a distinct row. Now, the problem is to compute the trace of the matrix, the trace of the matrix is the sum of the diagonal elements. So, it is defined as summation from i equal to 0 to n minus 1 of a sub i i. Notice that the matrix row indexing starts from 0, similarly the matrix column indexing also starts from 0.

(Refer Slide Time: 01:34)



```
# include <stdio.h>
main () {
        int i, j;       /* i runs over rows, j over cols */
        int n;          /* dimension n X n of the matrix */
        int a;          /* the current matrix entry read */
        float trace = 0.0;     /* sum of diag elements seen so far*/

        scanf("%d", &n);            /* read dimension of matrix */

        for(i=0;   i<n;   i++) { /* for each row do */
           for( j=0; j<n;  j++) { /* for each col of row i do */
               scanf ( "%d",   &a);     /* read A_ij into a */
               if ( i==j )
                  trace = trace + a;
           }
        }
}
```

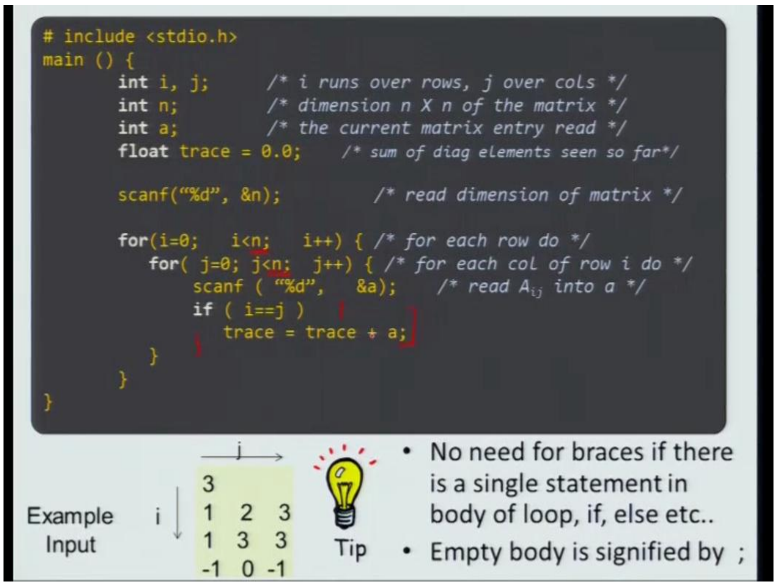
$$\begin{array}{ccc} \textcircled{1} & 2 & 3 \\ 4 & \textcircled{5} & 6 \\ 7 & 8 & \textcircled{9} \end{array} \quad \text{Trace} = 1+5+9$$

So, let us write a c program to solve this problem. Now, you should be familiar with how we compute the trace of a matrix. So, for example, if that matrix is given as let us say 1 2 3 4 5 6 7 8 9. So, the way we do it by hand is, look at the first row only this element goes into the trace. So, it is trace is 1 plus, no other element of the row goes into the trace, in the second row the second element goes into the trace. So, it is 5 plus and then no other remaining element goes into that trace and you go to the third row. And the third element goes in to the trace so, 1 plus 5 plus 9, this is how we do it by hand. You go row by row and then pick out for each row pick some element which goes into that trace only the diagonal element will go into the trace. Let us try to code this.

So, in this we have 2 variables i and j which I will use to iterate over the row indices and the column indices n is the designator for the size of the matrix. For example, the dimension of the matrix is n cross n. Now, a is the variable in to which I will read the current entry and then trace is the sum of the diagonal elements seen so far. I will assume that it is an integer matrix, it is not general enough you can use a float variable as well. I will first scanf the size of the matrix, n the matrix is of dimension n cross n. Once I have done that, here is what I was talking about in the previous slide, once you scan the number n you know that the matrix is n cross n. So, the number of times that you are going to iterate is known in advance. So, the number of times that you have to iterate is known before you enter the for loop. In such cases the for loop is more convenient to write than the while loop.

So, the outer loop is for each row from i equal to 0, to i equal to n excluding i equal to n you increment the row. Similarly, for j equal to 0 to n you increment the column index so, j is supposed to be the column index. Now, you scan the number a now if i equal to j remember that we wanted to add the only the diagonal elements. So, the diagonal elements will be when the row index is the same as the column index. So, when the row index is the same as the column index, you should add the corresponding numbers to the trace. So, once j becomes n minus 1, you will fail the test j is less than n. So, we will exit out of the inner for loop, and you will go to the outer for loop. In the outer for loop you have i iterating over the row indices. So, you will go to the next row and do the same processing for the next row, until you hit row index n at which point you will exit out of the outer for loop.

(Refer Slide Time: 05:23)



```
# include <stdio.h>
main () {
        int i, j;        /* i runs over rows, j over cols */
        int n;           /* dimension n X n of the matrix */
        int a;           /* the current matrix entry read */
        float trace = 0.0;    /* sum of diag elements seen so far*/

        scanf("%d", &n);          /* read dimension of matrix */

        for(i=0;   i<n;   i++) { /* for each row do */
           for( j=0; j<n;  j++) { /* for each col of row i do */
                scanf ( "%d",   &a);    /* read A_ij into a */
                if ( i==j )
                   trace = trace + a;
           }
        }
}
```

Example Input

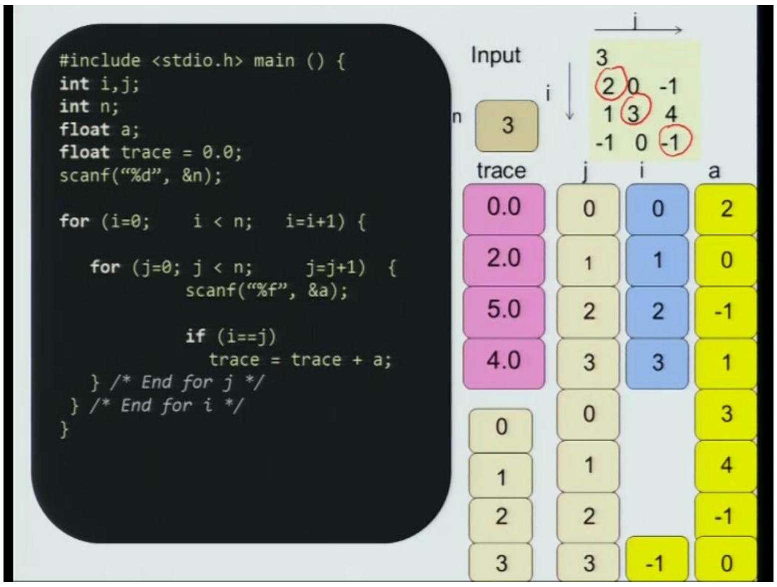
```
    j →
i ↓ 3
    1  2  3
    1  3  3
   -1  0 -1
```

Tip

- No need for braces if there is a single statement in body of loop, if, else etc..
- Empty body is signified by ;

So, let us look at a sample input let say that you have 1 2 3 1 3 3 and minus 1 0 minus 1. Here, is a particular convenience that c gives you which I have used in this code. So, notice that this if block I did not put the braces. So, it could have been necessary to put the braces according to the syntax that we have discussed so, far. But if there is only a single statement in the if block then, we do not need to put the braces and it is syntactically correct to do so.

(Refer Slide Time: 06:05)



```
#include <stdio.h> main () {
int i,j;
int n;
float a;
float trace = 0.0;
scanf("%d", &n);

for (i=0;    i < n;   i=i+1) {

   for (j=0; j < n;     j=j+1)  {
             scanf("%f", &a);

             if (i==j)
               trace = trace + a;
   } /* End for j */
 } /* End for i */
}
```

So, let us just run the program on a sample input. So, we have some sample array 2 0 minus 1 1 3 4 minus 1 0 1. So, initially there is this number 3. So, you know that it is a 3 cross 3 matrix. So, once you do that you know that n is 3 so, representing that it is a 3 cross 3 matrix. So, then you start with i equal to 0 and go on until i less than n incrementing i by 1 each time i is 0 i is less than n, because n is 3. So, you enter the outer loop the first statement of the outer loop is itself a for loop, you start with j equal to 0 j is less than 3. So, you enter the inner loop you scan a number a, which is a floating point number and if i equal to j. So, remember that we are looking for diagonal elements. So, we are currently at this point and i equal to 0 and j equal to 0.

So, we are entering we are scanning the zeroth element of the zeroth column of the zeroth row. So, that element has to go into the trace. So, i equal to j is true and then you say that trace equal to trace plus a trace was initialize to 0 so, trace becomes now 2. Once you do that, you iterate the inner for loop. So, you go to the updates statement in a inner for loop j becomes j plus 1. So, you go to the next column and the j is less than 3. So, you scan the next number 0 if i equal to j that is false now, because i is 0 and j is 1. So, you do not execute the if statement and go to the update statement. So, j becomes 2, 2 is less than 3. So, you scan 1 more number which is minus 1, i is not j. So, you update again j becomes 3, now 3 is not less than 3, so, you exit out of the inner loop. When you exit out of the inner loop there are no more statements to execute. So, you go directly to the update statement in the outer loop which becomes i equal to i plus 1. So, you are reading the first row, row number 1 you are finished reading row number 0. Again you scan the numbers when j equal to 1 that is the second number in the second row, you will see that i equal to j, because i is 1 and j is 1. So, you will add it to the trace. So, that is 2 plus 3 which is 5. So, trace gets updated and after you do that you scan the remaining entry in the same row, but it does not go to the trace, and then you have done with the row.

After that again you go to the outer loop you update the row index of the row index is less is 2 which is less than 3. So, you exit so, you enter the if condition and you execute the inner loop when i equal to 2 and j equal to 2 you will find an element which is minus 1 which will go in to the trace. So, the elements that will be added to that trace are when 2 3 and minus 1. Once you are done you get out of the inner loop and then you go into the outer loop and update it, but then i becomes 3 it is no longer true that 3 is less than 3. So, you have done reading all the rows. So, you exit the program when you exit the program you have the correct trace which is 4.