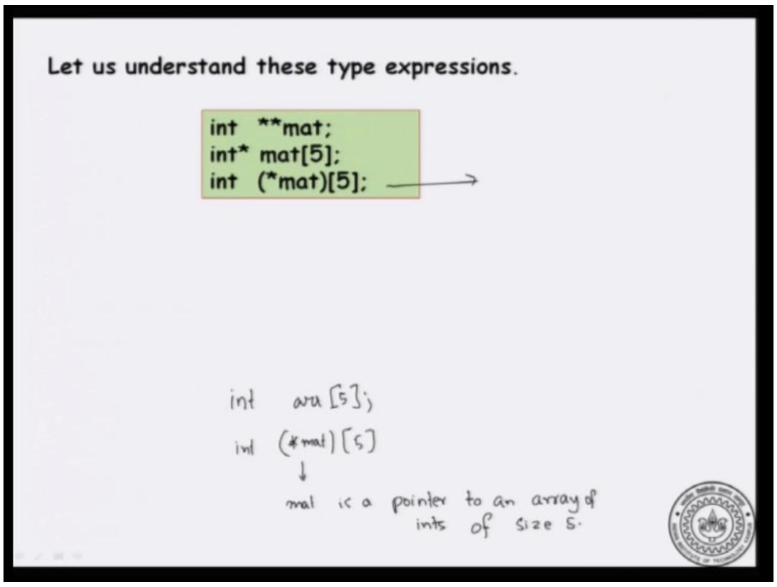
Introduction to Programming in C

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

Lecture - 45

In this video, we will look at one of the other expressions.

(Refer Slide Time: 00:03)

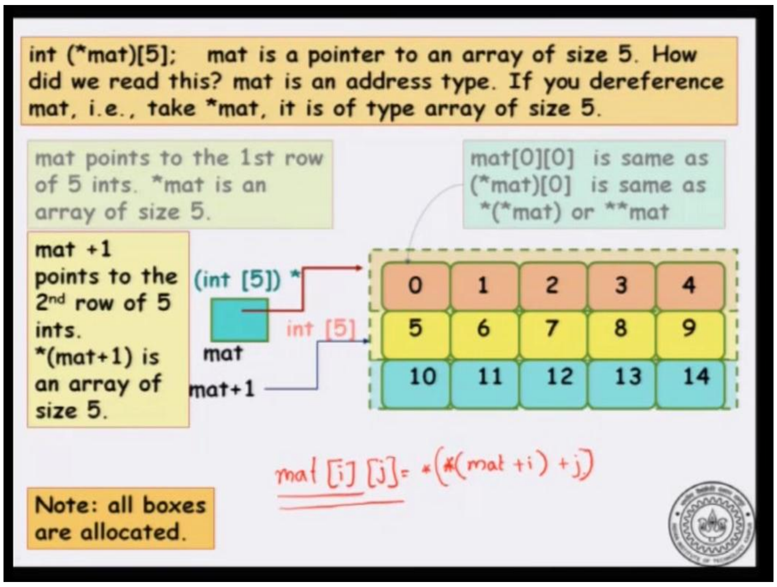


Let us understand these type expressions.

```
int  **mat;
int* mat[5];
int  (*mat)[5];
```

In particular, we will look at the third one, which is int star mat 5. So, if I had written int array 5, this means that array is an integer array of size 5. So, similarly I can read this as star mat is an integer array of size 5. So, in other words mat is a pointer to an array of size 5, array of ints of size 5. We can look at in this way and let us see, what this really means.

(Refer Slide Time: 00:54)



int (*mat)[5]; mat is a pointer to an array of size 5. How did we read this? mat is an address type. If you dereference mat, i.e., take *mat, it is of type array of size 5.

mat points to the 1st row of 5 ints. *mat is an array of size 5.

mat[0][0] is same as (*mat)[0] is same as *(*mat) or **mat

mat +1 points to the 2nd row of 5 ints. *(mat+1) is an array of size 5.

$mat[i][j] = *(*(mat+i)+j)$

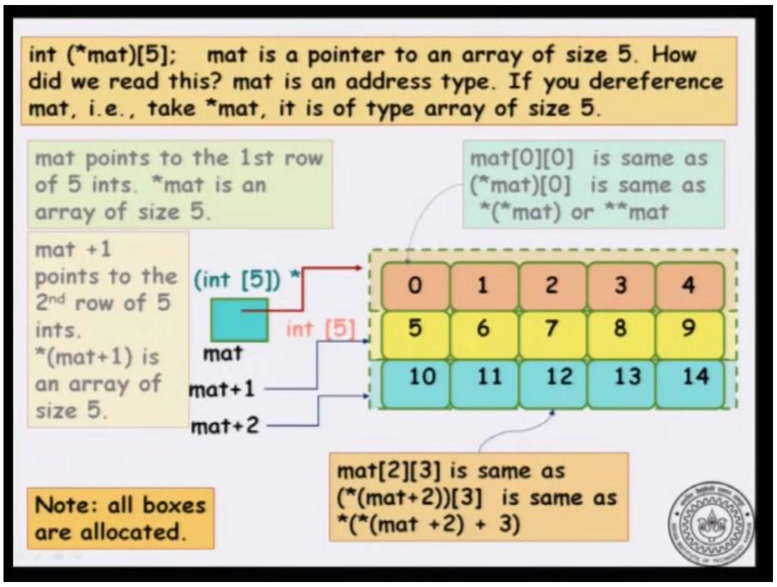
Note: all boxes are allocated.

So, we can picturize in this way, if you dereference mat that is, if you take star mat, you will get some array of size 5 of integers. Now, let us look at the pictures. So, mat may be pointing to some array of size 5, which means that the next subsequent location will be another array of size 5, if it is a valid address. Now, for the first location we can refer to it as mat 0 0 or it is the same as star mat of 0 or it is the same as star star mat.

So, remember the general formula that we had was, if I have the notation mat i j, I can look it up as star mat. So, first let me translate mat i,. So, that we have seen that this is simply dereferencing mat plus i, that address. So, now we have one more subscript. So, in order to decode that, I will do the formula for a second time, so, this plus j. So, remember that this is the general form. So, similarly if you have mat 0 0, I can write it as star mat of 0 or I can write it as star star mat, because i and j are both 0s.

So, this is just a special case of the general form, mat plus 1 points to the second row of 5 integers. So, remember that the type of mat is, it is a pointer to an array of size 5 of integers. So, the next pointer location when you do mat plus 1 goes to the next array of size 5. So, mat plus 1 is another array of size 5. In particular, it may be the second row of a two dimensional array, where you have 5 columns, mat plus 2 will be similarly the third row and so, on.

(Refer Slide Time: 03:10)



int (*mat)[5]; mat is a pointer to an array of size 5. How did we read this? mat is an address type. If you dereference mat, i.e., take *mat, it is of type array of size 5.

mat points to the 1st row of 5 ints. *mat is an array of size 5.

mat[0][0] is same as (*mat)[0] is same as *(*mat) or **mat

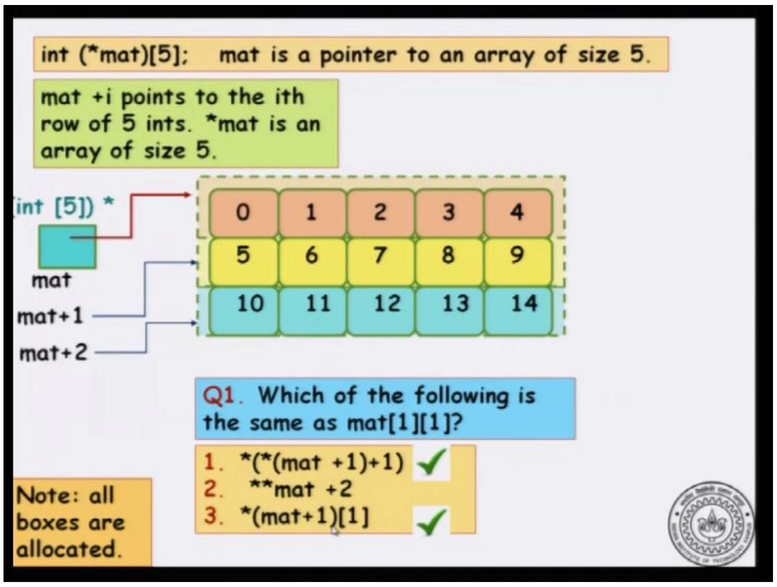
mat +1 points to the 2nd row of 5 ints. *(mat+1) is an array of size 5.

mat[2][3] is same as (*(mat+2))[3] is same as *(*(mat +2) + 3)

Note: all boxes are allocated.

So, mat 2 3 for example, if you apply the formula, it will come out to be star of star of mat plus 2 plus 3. Notice that, all boxes are allocated in this example.

(Refer Slide Time: 03:25)



int (*mat)[5]; mat is a pointer to an array of size 5.

mat +i points to the ith row of 5 ints. *mat is an array of size 5.

Q1. Which of the following is the same as mat[1][1]?

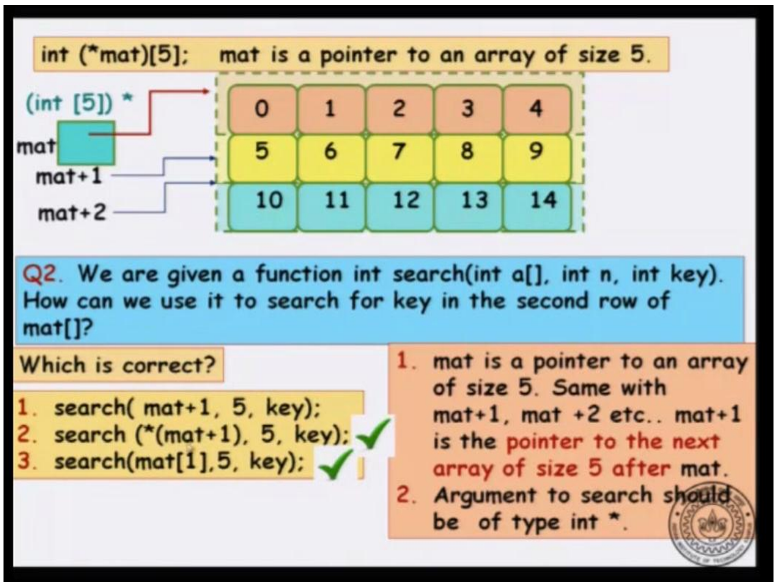
1. *(*(mat +1)+1) ✓
2. **mat +2
3. *(mat+1)[1] ✓

Note: all boxes are allocated.

Now, mat plus i points to the i th row of 5 integers and star mat is an array of size 5, this is what we have seen. Now, you can in order to get comfortable with a notation, you can look at these formulas and try to decode. Like for example, you could try, what is the arithmetic way of representing the location mat 1 1. So, you can see that it is definitely the first case, where it is star of star mat plus 1. So, that is definitely true, because this is just the formula that we just now discussed.

But, if I do not decode both the subscripts, I decode only one subscript using pointer arithmetic and leave the other subscript as it is, then I know that it is also equivalent to 3. So, 3 is also another way of representing it and tried to convince yourself, why the second is not correct?

(Refer Slide Time: 04:28)



int (*mat)[5]; mat is a pointer to an array of size 5.

Q2. We are given a function int search(int a[], int n, int key). How can we use it to search for key in the second row of mat[]?

Which is correct?

1. search(mat+1, 5, key);
2. search (*(mat+1), 5, key); ✓
3. search(mat[1],5, key); ✓

1. mat is a pointer to an array of size 5. Same with mat+1, mat +2 etc.. mat+1 is the pointer to the next array of size 5 after mat.
2. Argument to search should be of type int *.

Now, let us understand this in somewhat more detail by considering a tricky question, we have a function int search. So, here is a function int search, int a, int n, int key. So, what does this function do? It will search for key inside array a of size n, a is an array with n elements and you have to search for it, search inside for it for the element key. If it is found, then you return the index where it is found, if it is not found, you return minus 1.

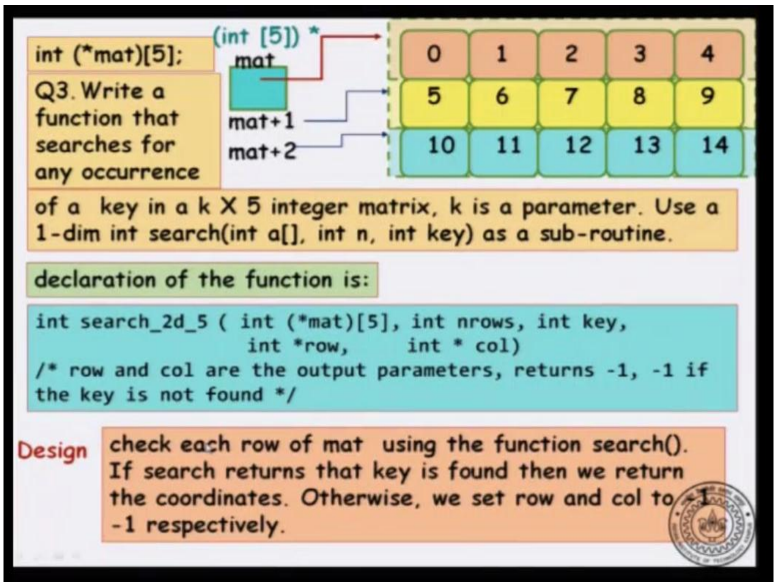
Because, minus 1 can never be a valid index in an array. So, when you return minus 1, you know that it is not present in the array. Now, can we use this, a function to search inside a 2D array. So, we are using a one dimensional function, in order to search inside a 2D array. Now, the basic idea is that we can search row by row, each row of a two dimensional array is somewhat like a one dimensional array. So, we will call search multiple times, once for each row in the array, until we either find it or we are done with all rows. The algorithm is, search it row by row.

Now, the question is which of the following is actually doing that? So, we have three expressions, search mat plus 1 5 key, search star of mat plus 1 5 key and search mat of 1 5 key, which of these will do it. Now, let us look at second, mat is pointing to an array of size 5. Therefore, mat plus 1 is also a pointer to an array of size 5, when we dereference that, we get an array of size 5,. So, that is the right type.

So, the first argument to search the second statement will be an array of size 5. So, therefore, the second call is valid. What about the third call? Again, mat of 1 is simply star of mat plus 1, if you translated into pointer arithmetic. So, the third line is just the second line in discussed, instead of using pointer arithmetic notation, we are using subscript notation so, 2 and 3. In fact, are equivalent, so, 2 is correct. Therefore, 3 is also correct.

Now, think about why statement 1 does not make sense. So, mat plus 1 is actually a pointer to an array of size 5. Therefore, it is not the right type, it is not an array of size 5, it is a pointer to an array of size 5. So, it is not the correct type and therefore, the first call is not valid, the first option is a big delicate. So, I would encourage you to stop here and think about, why it is not correct?

(Refer Slide Time: 07:37)



int (*mat)[5];

Q3. Write a function that searches for any occurrence of a key in a k X 5 integer matrix, k is a parameter. Use a 1-dim int search(int a[], int n, int key) as a sub-routine.

declaration of the function is:

```
int search_2d_5 ( int (*mat)[5], int nrows, int key,
                  int *row,     int * col)
/* row and col are the output parameters, returns -1, -1 if
the key is not found */
```

Design

check each row of mat using the function search(). If search returns that key is found then we return the coordinates. Otherwise, we set row and col to -1 -1 respectively.

Now, let us utilize the function in order to write our routine to search inside a 2D array. So, once again we are utilizing a one dimensional search routine in order to search inside a two dimensional array. So, let us say that, we are given this int search function which can search inside a one dimensional array for a key. Now, I will write a 2D function, a function which can search inside a 2D array.

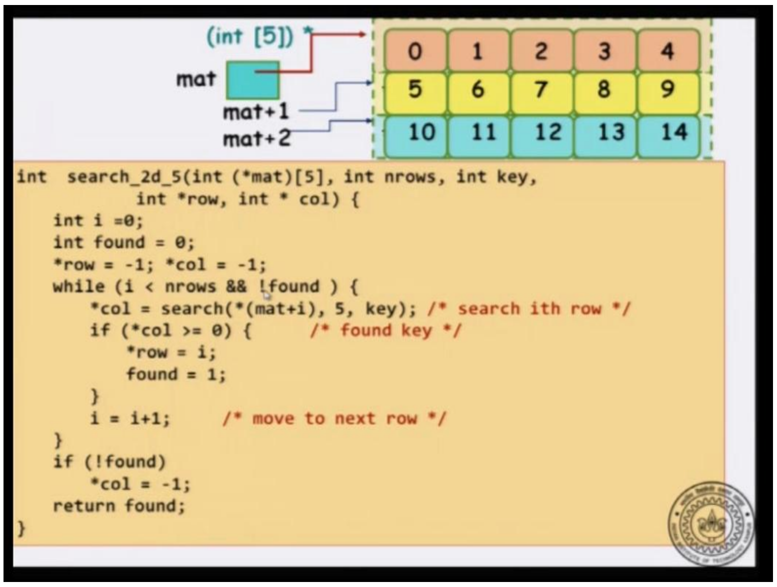
Now, the correct declaration of the function would be int star mat 5, int n rows int key, n rows is going to be the number of rows in the array. Key is the key, we are searching for and int star row and int star column. So, I want to focus on the first argument and the last two arguments. The first argument says that, I will pass you a pointer to an array of size 5, this is exactly what we should do because, then a two dimensional array can be just traverse by using mat plus 1, mat plus 2 and so, on.

So, here is the correct type declaration that should accompany the 2D search routine, nrows is just the number of rows, key is the key. Why are we saying, int star row and int star column? We want to return two things, if a key is found, we want to return it is row index and it is column index. Now, unfortunately a function can return only one value. So, how will you return two values?

So, we will say that we will not return two values. What we will do is, give me a pointer and I will write in that address, the correct row and the correct column, if it is found. Here is a standard way in C, where you might encounter a situation where you need to return two values and instead, what you pass are the pointers. The algorithm is what we have discussed before. You check each row of mat using the function search. If search returns success, then that will be the column index in that row, because search is searching inside a 1D array.

So, wherever it returns that will be the column index in the i th row. So, now you say that the column index is that and the row index is the i that I had. If it is not found in any of the rows, you return minus 1.

(Refer Slide Time: 10:16)



```
int  search_2d_5(int (*mat)[5], int nrows, int key,
          int *row, int * col) {
    int i =0;
    int found = 0;
    *row = -1; *col = -1;
    while (i < nrows && !found ) {
        *col = search(*(mat+i), 5, key); /* search ith row */
        if (*col >= 0) {      /* found key */
            *row = i;
            found = 1;
        }
        i = i+1;      /* move to next row */
    }
    if (!found)
        *col = -1;
    return found;
}
```

So, let us write the function, we have an i to go traverse for the rows, we have found equal to 0, this will be the flag indicating whether the key is found or not. And initially, you just set star row equal to minus 1 and star column equal to minus 1 to indicate that I am not yet found it, found the key. Now, you write the main loop which is going through the rows one by one. You start with row 0 and you go on, until both these conditions are true. That is, you have not seen all the rows, i is less than n rows and you have not found the key, so, not found.

What should you do to the i th row? I should say that search the i th row. So, the way I say it is, search star of mat plus i. This is the same as saying search mat of square bracket i comma 5, which is the number of columns and key, which is the key that I want to search for, the return value is stored in star call. So, you dereference call and store the return value there. Now, search can return either you if the key is found, it will return the correct column index or it will return minus 1.

So, you just check for that, if star column is a non-negative number, then you say that it has been found. So, you say that the row is i, So, star row is i and found is now 1. So, at the next iteration you will exit out of the loop, because you have found the key. And then, the last statement in the loop will be just to increment the i variable. Finally, if you have done with all the rows and if you have exited out of the while loop, you check whether you exited out of the while loop, because you exhausted all the rows.

So, there are two conditions to exit the while loop, one is i is greater than or equal to n rows, that is one condition. The second is that found equal to 1, if you exited because, found equal to 1, then you can return the correct value without any problem. If you exited before, if all the rows were exhausted and you still did not find the key, then you have to say that column is minus 1. So, here is a brief code which will do this.

So, this code utilizes our understanding of two dimensional arrays as basically an pointer to an array of size 5 and here is why the number of columns is important. Because, in order to do mat plus 1 correctly, we need to know how many bytes to skip and this is crucially depended on the number of columns. The number of rows actually does not matter. Because, you can keep on incrementing the rows as long as the array is valid. The number of columns is important, because that is how you get to the next row.