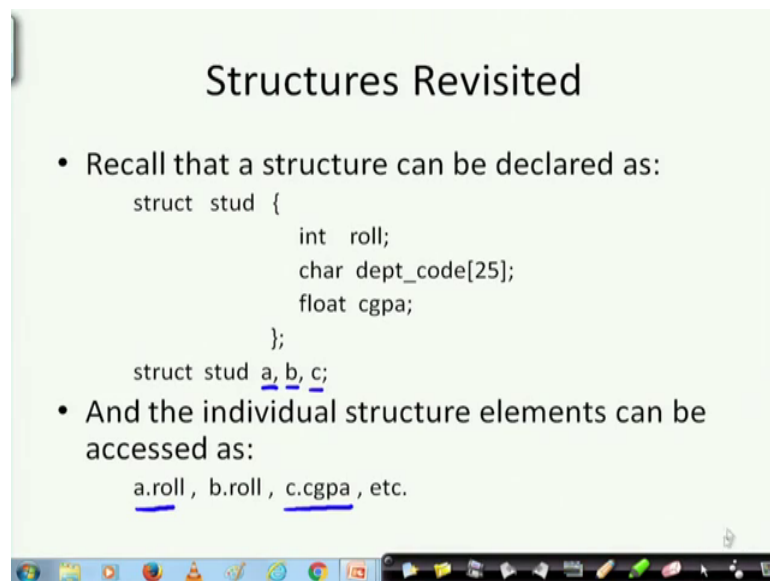


Problem Solving through Programming In C
Prof. Anupam Basu
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture – 60
Pointer in Structures

We have seen how structures are represented, we have also learnt about pointers.

(Refer Slide Time: 00:25)



Structures Revisited

- Recall that a structure can be declared as:

```
struct stud {  
    int roll;  
    char dept_code[25];  
    float cgpa;  
};  
struct stud a, b, c;
```
- And the individual structure elements can be accessed as:
a.roll , b.roll , c.cgpa , etc.

So, today we will be looking at the structures once again in a different light as you can see here, a structure can be declared as is shown here the student structure struct stud is consisting of three members or three fields.

So, they are the role, the department code which is a character array and cgpa which is a floating point number. And a b c are three variables of the type s t u d stud. The individual structure elements you now can be accessed by a dot role. So, a is this field or b dot role c dot c g p a. So, with this dot operator we can access them this was known to us right.

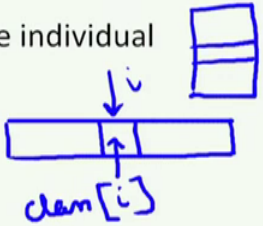
(Refer Slide Time: 01:19)

Arrays of Structures

- We can define an array of structure records as

```
struct stud class[100];
```
- The structure elements of the individual records can be accessed as:

```
class[i].roll  
class[20].dept_code  
class[k++].cgpa
```



The diagram shows a horizontal array labeled 'class' with several cells. A blue arrow labeled 'i' points to one of the cells. Below the array, the text 'class[i]' is written in blue. To the right of the array, a vertical stack of three boxes represents a structure record, with a blue arrow pointing from the selected cell in the array to the top box of the structure.

Now, we can also we have seen that, we can define an array of structures where class is an array of students, class sizes 100 and each element is a structure of type stud. The structure elements of the individual records can be accessed with this dot operator like class i any particular element of that say for example, here, any particular element I take and.

So, this is class i where i is this index I come over here, and get a particular field of this. So, this has got a number of fields. So, I am coming to the role field or the department code field dot the cgpa field this was also known to us.

(Refer Slide Time: 02:18)

Example: Sorting by Roll Numbers

```
#include <stdio.h>
struct stud
{
    int roll;
    char dept_code[25];
    float cgpa;
};

main()
{
    struc stud class[100],t;
    int j, k, n;

    scanf ("%d", &n);
    /* no. of students */

    for (k=0; k<n; k++)
        scanf ("%d %s %f", &class[k].roll,
            class[k].dept_code, &class[k].cgpa);
    for (j=0; j<n-1; j++)
        for (k=j+1; k<n; k++)
        {
            if (class[j].roll > class[k].roll)
            {
                t = class[j];
                class[j] = class[k];
                class[k] = t;
            }
        }
    <<<<PRINT THE RECORDS >>>>
}
```

Handwritten annotations:
- "Array is read" with a bracket pointing to the `scanf` loop.
- "Swapping structure" with a bracket pointing to the `if` block.
- A diagram at the bottom shows an array with elements at indices `k` and `j` being swapped.

So, here we are trying to apply that to an example sorting by roll numbers. So, here we have got a set of students look at the declaration here student or struct stud is the structure, having role department code and cgpa. Now, in the main function what are you doing? We are defining struct student class 100. Now, struct start has been declared, it has been declared globally even before main that is possible ok.

And here I am saying that class 100 is one array and t and there are some integers t is again a structure, t is also a structure of type stud. Now, I am reading the number of students how many students are there in the class. The array can accommodate at most 100 but I am reading the value n. Now, for each of the elements, look at this is our familiar for loop here for k is equal k 0 to less n, I am scanning the class the roll number of that particular student the department code and the c g p.


So, in that way I read here I am reading the array all right. The array is being read here array is read at this point right. Now what I am doing now? Next let us look at this loop what are you doing here? For some value of j if the role of that student; So, here I have got my array, I am coming to any particular j a particular element and looking at the role number field of that, if that is greater than some other value k if is greater than the roll value of k, then I am exchanging them just as we sort.

Now, what are we sorting here in that example earlier that we had done, we are sorting integers or sorting real numbers here what we are doing we are sorting the entire structure as you can see here the entire structure the j is coming to t.

So, t is again a structure k is coming to j and so, this is a swap operation swapping structures swapping the structure structures all right and this I am doing for k is a internal variable and j is the external variable. So, my diagram should be a little different the diagram should be.

(Refer Slide Time: 05:28)

Example: Sorting by Roll Numbers

<pre>#include <stdio.h> struct stud { int roll; char dept_code[25]; float cgpa; }; main() { struc stud class[100],t; int j, k, n; scanf ("%d", &n); /* no. of students */</pre>	<pre>for (k=0; k<n; k++) scanf ("%d %s %f", &class[k].roll, class[k].dept_code, &class[k].cgpa); for (j=0; j<n-1; j++) for (k=j+1; k<n; k++) { if (class[j].roll > class[k].roll) { t = class[j]; class[j] = class[k]; class[k] = t } } } <<< PRINT THE RECORDS >>></pre> 
---	--

So, for every j I am looking at one j and I am varying k from here this is k, k is varying from this point to this point to the end of the structure and comparing with this j value. If this is any element that is less that is coming over here and it is being swapped. So, you have seen this swap algorithm earlier.

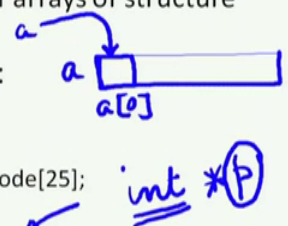
So, that is being swapped here and then I update j and I go on doing this. So, this is how I can apply the sorting algorithm that we had learnt, that can be applied for sorting student records all right.

(Refer Slide Time: 06:20)

Pointers and Structures

- You may recall that the name of an array stands for the address of its zero-th element.
 - Also true for the names of arrays of structure variables.
- Consider the declaration:

```
struct stud {  
    int roll;  
    char dept_code[25];  
    float cgpa;  
} class[100], *ptr;
```



Next the other thing that we have to look at is pointers and structures how they can be intermingled? You may recall that the name of an array stands for the zeroth element of the array. So, if this be an array then if the name of the array `a` is name of the array is `a`, then the name `a` and `a[0]` are synonymous and so, `a` is also `a` can become considered to be a pointer that is pointing to `a[0]` all right.

Now, consider the declaration this, there is also true for the names of arrays of structures. So, if I have a declaration like this, you look at the declaration yourself. So, you can see `stud` is a structure and I have declared of the type `stud` and array `class` 100 and star `ptr`.

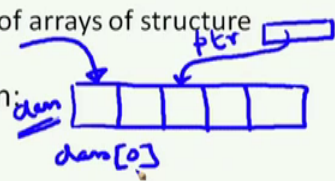
Now, what is star `ptr` what does it mean? You know by now just as we had done this keep it side by side `int star p` what does it mean? It means that `p` is a pointer that points to integer data type. Here it means that `ptr` is a pointer that points to start type of structures, this type only to this type of structure `ptr` points to that. So, now, what do I have?


(Refer Slide Time: 07:59)

Pointers and Structures

- You may recall that the name of an array stands for the address of its zero-th element.
 - Also true for the names of arrays of structure variables.
- Consider the declaration:

```
struct stud {  
    int roll;  
    char dept_code[25];  
    float cgpa;  
} class[100], *ptr;
```





I have got an array called class each element of which is of type student structure and I have got a pointer ptr, which can point to such structures ok. It can point to such structures any of these I have not yet initialized this.

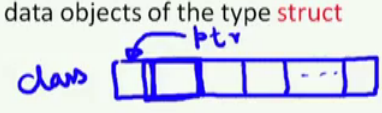
(Refer Slide Time: 08:29)

- The name **class** represents the address of the zero-th element of the structure array.
- **ptr** is a pointer to data objects of the type **struct stud**.

- The assignment

```
ptr = class;
```

 will assign the address of **class[0]** to **ptr**.
- When the pointer **ptr** is incremented by one (**ptr++**):
 - The value of **ptr** is actually increased by **sizeof(stud)**.



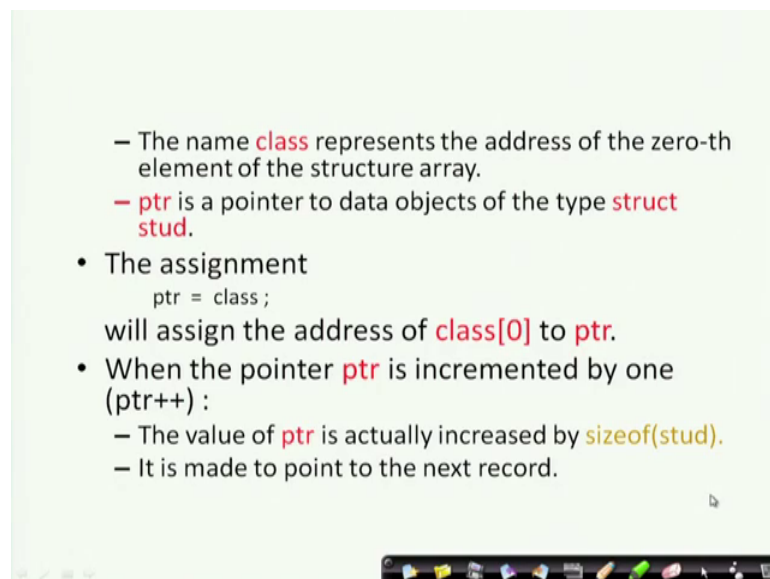
The name class therefore, represents the address of the zeroth element of the structure array. So, if we go back. So, if I have this then this class means it is a pointer that is pointing to this element plus 0 all right just like an array. So, and ptr is a pointer to the data objects of type struct that we have seen.

The assignment `class` assigned to `ptr` what will it make? It will make `ptr` now to point to the first element of the array here is `class` and `ptr` is now pointing, when I do this assignment note that they are of the same type. So, when I do this assignment, this is pointing to this element all right.

So, it will assign `ptr` to `class[0]` understood? When the pointer `ptr` is incremented by 1 that is `ptr++` you should be able to tell me what will happen. The `ptr` will be incremented to the next element of the array. So, the actual increment will be by a scale factor and what is that scale factor? That scale factor is size of `stud` size of the structure `stud` and you know that it has got a roll number, it has got `c g p a`.

So, depending on the different data types that are housed inside that structure it will vary. So, the value of `ptr` will point to the next element will be incremented by size of `stud` fine this much is clear.

(Refer Slide Time: 10:49)



- The name `class` represents the address of the zero-th element of the structure array.
- `ptr` is a pointer to data objects of the type `struct stud`.
- The assignment
`ptr = class;`
will assign the address of `class[0]` to `ptr`.
- When the pointer `ptr` is incremented by one (`ptr++`):
 - The value of `ptr` is actually increased by `sizeof(stud)`.
 - It is made to point to the next record.

(Refer Slide Time: 10:51).

- Once `ptr` points to a structure variable, the members can be accessed as:
`ptr->roll;`

class [1]. cgpa
ptr = class [0]; cgpa
ptr++
ptr -> cgpa

class [0]
class [1]
ptr -> cgpa;

So, its permit to point to the next record; once `ptr` points to a structure variable now this is something new the pointers can be accessed as `ptr` role that is possible. So, let us see what is happening you are being introduced to this operator the arrow operator.

So, here or this only comes if the left side of this operator is a pointer. So, it is if suppose I have got some structure and pointer `p` or `ptr` is pointing to this structure and it has got different fields say `cgpa` is a field. If I write `ptr -> cgpa`; that means that I am now pointing to this I am actually accessing this element of the structure.

Just as suppose this structure is `class` and this element is `class 1` say. I could have written `class 1 . cgpa` and here what I have done is I have already done this that `ptr`, the pointer `ptr` was assigned to `class 0` and then I did `ptr++`; that means, now where is `ptr` pointing to? It is `ptr` is pointing to `class 1`. So, I can also do `ptr -> cgpa` these two are equivalent this and this are equivalent.

So, here also you can understand this. So, let us move ahead.

(Refer Slide Time: 13:11)

- Once **ptr** points to a structure variable, the members can be accessed as:
`ptr -> roll ;`
`ptr -> dept_code ;`
`ptr -> cgpa ;`
- The symbol “->” is called the **arrow** operator.

Similarly, I can go to another field by ptr slash department code, I can go ptr cgpa, ptr roll the symbol this symbol is naturally called the arrow operator example.

(Refer Slide Time: 13:31)

Example

```
#include <stdio.h>

typedef struct {
    float real;
    float imag;
} COMPLEX;

swap_ref(COMPLEX *a, COMPLEX *b)
{
    COMPLEX tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}

print(COMPLEX *a)
{
    printf("(%.2f, %.2f)\n", a->real, a->imag);
}
```

You can read this type def now here I am defining a type the name of the type is complex and what is this type complex it is a structure with real and imaginary parts this is what we have what we saw earlier right.

Next I have got a function print complex star a what does it mean? A is a pointer to type complex. So, print f whatever there are two two placeholders a real a imaginary. So, why

is it possible? A is a pointer. So, it is pointing to some complex number, it is pointing to some complex number some complex number whatever that is a is a pointer pointing to that and I am going to the real part I am printing a real and this one this part is a imaginary. So, this is what is being done by this piece of function code.

So, now swap reference, I have got complex a and complex b that is now I am sorry it is not being visible here there are two complex a and b there are two pointers in complex. So, there is t m p I take another structure of named t m p of type complex, and I swap the pointers here.

That is why you can you can work it out that; that means, that a was pointing to some structure and that that a is pointing to this structure and this particular structure is copied to another structure temp, and b then b was pointing to another structure b and a are swapped. So, now, b is pointing to this and a is pointing this next I am making b to point to this. So, that is how a swap operation can be done and how we passed on the parameter to this.

(Refer Slide Time: 16:01)

```
Example

#include <stdio.h>

typedef struct {
    float real;
    float imag;
} COMPLEX;

swap_ref(COMPLEX *a, COMPLEX *b)
{
    COMPLEX tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}

print(COMPLEX *a)
{
    printf("(%.6f,%.6f)\n",a->real,a->imag);
}

main()
{
    COMPLEX x={10.0,3.0}, y={-20.0,4.0};
    print(&x); print(&y);
    swap_ref(&x,&y);
    print(&x); print(&y);
}

(10.000000,3.000000)
(-20.000000,4.000000)
(-20.000000,4.000000)
(10.000000,3.000000)
```

So, the main program can be can define complex x, 10 and 3 y is minus 20 minus 10 because of the resolution it is not visible. So, print x and y then swap reference print x and y. Now, can you guess and tell me whether the swapping will work in this case or not the answer is yes it will work because it is a call by reference. So, here I whatever I am

swapping here, that will be reflected in the main program also. So, here is an example. So, let us move ahead.

(Refer Slide Time: 17:04)

A Warning

- When using structure pointers, we should take care of operator precedence.
 - Member operator “.” has higher precedence than “*”.
 - $ptr \rightarrow roll$ and $(*ptr).roll$ mean the same thing
 - $ptr.roll$ will lead to error.
 - The operator “->” enjoys the highest priority among operators.

Now, one warning is there that, when using a structure pointers we should take care of the operator precedence. Because here we have seen the star symbol as well as dot symbol right; we have seen the star symbol as well as the dot symbol, now this dot has a higher precedence than star what does it mean?

It means that if I have got something like $ptr \rightarrow roll$, then and $star \ ptr \ dot \ roll$ what will happen this dot has got a higher precedence. So, these two are the same thing. $ptr \rightarrow roll$ ptr is a pointer and the $roll$ is that particular field and $star \ ptr$ is what? $Star \ ptr$ is the $star \ ptr$ is the that particular structure, and I am going to the $roll$ field of that structure.

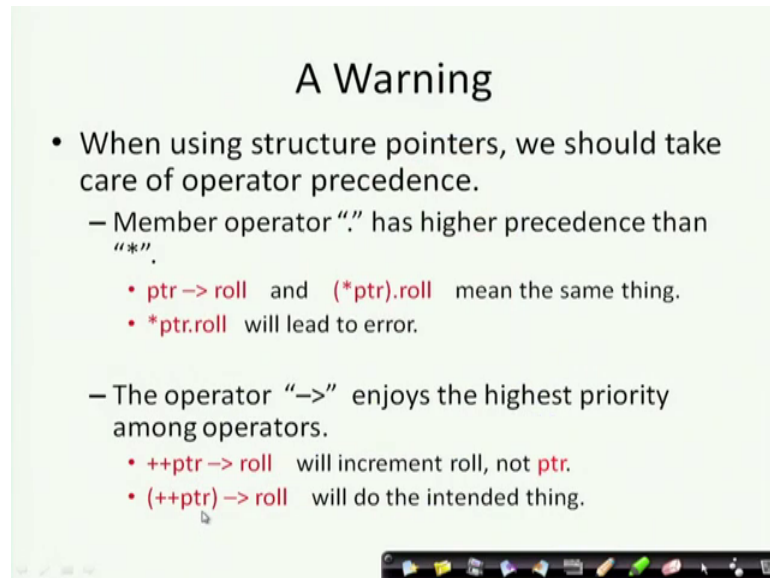
So, $star \ ptr \ dot \ roll$ that is these two are equivalent, but if I had done this that would be an error why? That would be an error because this dot operator has got higher precedence than this star operator. So, basically I will try to get the role filled from the pointer $p \ t \ r$, but that is not the case, I cannot get that there is no $roll$ field inside the pointer $p \ t \ r$. The $roll$ field is there in the content of the pointer ptr that is $star \ p \ t \ r$.

So, I must make first $star \ ptr$ within bracket then dot $roll$. So, this is correct to mean this, but this is wrong the and the operator arrow enjoys the highest priority among operators among all these operators it has got the highest priority.

(Refer Slide Time: 19:12)

A Warning

- When using structure pointers, we should take care of operator precedence.
 - Member operator “.” has higher precedence than “*”.
 - `ptr->roll` and `(*ptr).roll` mean the same thing.
 - `*ptr.roll` will lead to error.
 - The operator “->” enjoys the highest priority among operators.
 - `++ptr->roll` will increment roll, not `ptr`.
 - `(++ptr)->roll` will do the intended thing.



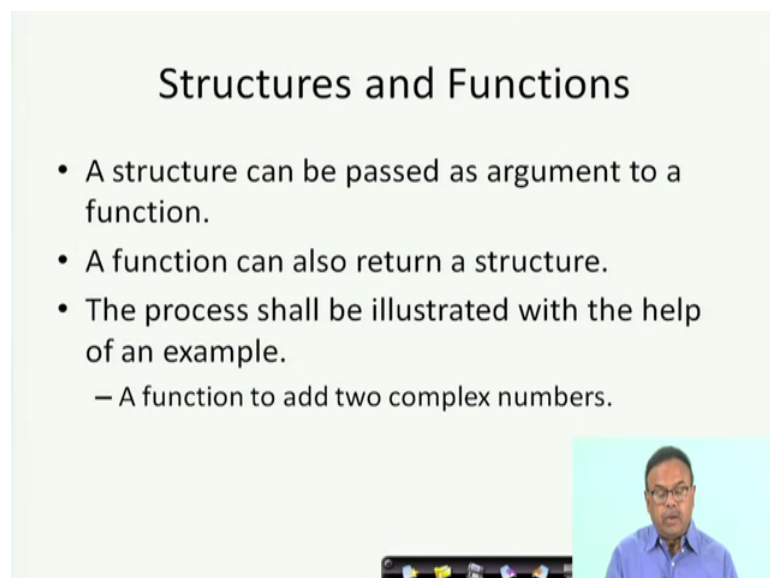
So, we have learnt. So, here in this case what will happen plus plus ptr arrow roll as we have said that this has got the highest priority we will go to where the ptr is and go to the roll field of that and after that we will increment this plus plus although it is shown as a pre increment. So, that will not have higher precedence than the arrow operator.

So, if I want to first increment the ptr and then get the roll, then I should do plus plus ptr arrow roll will do the intended thing, this will not do the intended thing I hope this is clear this part.

(Refer Slide Time: 20:00)

Structures and Functions

- A structure can be passed as argument to a function.
- A function can also return a structure.
- The process shall be illustrated with the help of an example.
 - A function to add two complex numbers.



Now, structures and functions. Structures can be passed as argument to a function and a function that we have already seen a function can also return a structure. So, we have seen that using two complex numbers how we can do that.

(Refer Slide Time: 20:23)


Example: complex number addition

```
#include <stdio.h>
struct complex {
    float re;
    float im;
};

main()
{
    struct complex a, b, c; ✓
    scanf ("%f%f", &a.re, &a.im);
    scanf ("%f%f", &b.re, &b.im);
    c = add (a, b); ✓
    printf ("n %f %f", c.re, c.im);
}
```

```
struct complex add (x, y)
struct complex x, y;
{
    struct complex t;

    t.re = x.re + y.re;
    t.im = x.im + y.im;
    return (t);
}
```



So, here again we look at that example that we have got a complex number, complex structure with two fields I have now changed the names float r e and float I m, and in the main function I am scanning and a dot r e real part of a real imaginary part of a, then real part of b then imaginary part of b, and then I call the function add a b. What is happening here when I call this add a b, what am I passing a b and here x y.

So, it is a call by value right. So, here this is copied; now struct complex x y there is no problem here it is not like. So, up I just want to add. So, I have got two internal variables x and y and t that new structure t is taking xs real part and ys real part, then its I am getting the real part of t and in the imaginary part of t there is nothing here there is an excess space, in the imaginary part of t I take the imaginary part of x and the imaginary part of y, then I am returning this t to c, where c is also of type complex since the types are matching I can pass it on over there.

Now, this is one way of adding the complex numbers. Now if this is clear now let us look at an alternative way of using the pointers.

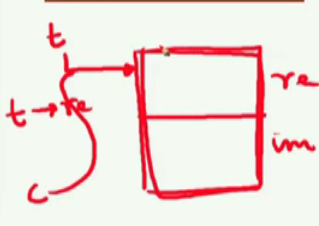
(Refer Slide Time: 22:17)

Example: Alternative way using pointers

```
#include <stdio.h>
struct complex {
    float re;
    float im;
};

main()
{
    struct complex a, b, c;
    scanf ("%f%f", &a.re, &a.im);
    scanf ("%f%f", &b.re, &b.im);
    add (&a, &b, &c);
    printf ("\n %f %f", c.re, c.im);
}
```

```
void add (x, y, t)
struct complex *x, *y, *t;
{
    t->re = x->re + y->re;
    t->im = x->im + y->im;
}
```



Here I just changed this other things are same. What I am doing here is I have called them now passing the address, now I am passing the pointers in the earlier case I was passing the value here. And now I am passing the pointers, I am passing address of a, address of b and address of c. Now, inside this add function what am I doing? I am taking x and y. So, these are pointers I am taking them and t, t is real. So, t is a type pointer so, t sorry.

So, t is a pointer that is pointing to another structure of the type complex and t is r e t arrow r e means a real field of this t, will be taking the real part of x c x and real part of y. So, here x plus y the sum will come, this the real part of x and real part of y will come here, and here the imaginary part here the imaginary part of x and y will be added ok.

Now, when here I am not returning any t, because this t is nothing, but this c; so, this t and c are mapped it is a call by reference. So, c is also actually pointing over here. So, I get the result here. So, this is another way in which in structures we can use pointers also. So, these are relatively more advanced aspects and you gradually practice this and we will get familiar.

Thank you.