

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

**Lecture – 40**  
**Parameter Passing in Function Revision**

So, we had looked at parameter passing and we have looked, we have seen that the difference between call by value and call by difference and we have see quite a few examples to be specific 3 examples on call by value how. And also another thing that was supposed to be noted I hope you have noted that, that is the scope of variables. That whenever there is an x in the main function and the x in the called function then these two x's are different. The x that is defined in the called function is a separate location than the x in the main function and that the life of that variable ends with the end of the function, right.

(Refer Slide Time: 01:15)

**Parameter passing & return: 4**

```
void main()
{
  int x=10,y=5;
  printf ("M1: x=%d, y=%d\n", x, y);
  interchange (x, y);
  printf ("M2: x=%d, y=%d\n", x, y);
}

void interchange (int x, int y)
{
  int temp;
  printf ("F1: x=%d, y=%d\n", x, y);
  temp= x; x= y; y= temp;
  printf ("F2: x=%d, y=%d\n", x, y);
}
```

**Output**

```
M1: x = 10, y = 5 ✓
F1: x = 10, y = 5
F2: x = 5, y = 10
M2: x = 10, y = 5
```

How do we write an interchange function?  
(will see later)

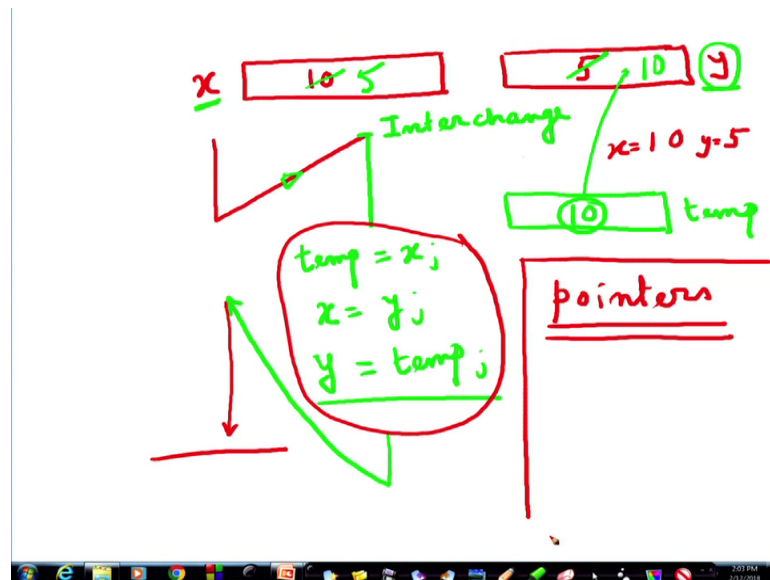
Now, let us look at these example, you yourself will be able to trace this. So, I give you some time to trace this through and then we will continue. Look at this function carefully and do not look at this green part, do not look at the green part yourself and try to without looking at that try to find out what the values will be for the different printf statements.

Let us start. So, we have got x assigned to 10 and y assigned to 5, x and y, x is 10 and y is 5. Now, so when the I printf do this printf x is printed to be 10 y is printed to be 5 fine, no issue. Then I call a function interchange x y. Look at this the type of this function interchange the type of this function interchange is void because that immediately tells us that it is not going to return anything it is going to do something as the name implies it is going to interchange x and y. And what are the parameters? Parameters are x and y this is the argument. So, here I have got another set of x and y.

Again note that this x and y are different. Now I initialize temp, so temp is another variable here temp. Now, what I am doing here? The first printf what is being done here printf x equal to x and y equal to y which x and which y the x this x and this y. Now, when I entered this function these values have been copied here. So, the printf will simply print 10 and 5 no issue now temp is getting x, x was 10. So, I put 10 here and then x assigned y sorry y assigned x; that means, the value of y I am still within the function I am within the function. So, this value is being assigned. So, by this statement what happens this becomes 5 and this remains 5 right and then what is being done here y assign temp; that means, this temp is becoming 5 right temp is becoming sorry, sorry I am sorry absolutely sorry this was 10.

Now, for this statement temp is going to y; that means, y will be changed to 10. So, here I find 5 and 10. So, it is a when it entered here I could see 10 and 5 and so when I come and print from here x will be 5 and y will be 10, I come out of the program and printf x and y. What will be printed? My god, what I find is x is being printed as 10 and y is being printed as 5 as it was here. So, no change has actually taken place. Why did it happen like this? You can immediately see the reason that whatever change took place took place here inside the function and this actually tells you the importance of the scope of variables. These variables scope ended with this function and their change was not reflected in the main here that is why although I did it here it would not change. However, suppose let us do some intellectual exercise here.

(Refer Slide Time: 06:43)



Suppose x was 10 and y was 5 and there was a temp, now temp is inside the function. So, I will use different colours for that I will, now although I remind you that C does not allow call by reference, but just to see whether you have understood call by reference properly. Suppose we are allowed to use call by reference then my main program has got this x and y. So, it comes here prints say prints x and y. So, 10 and 5 are printed x equal to 10 y is equal to 5 and it calls interchange all right, it calls interchange.

And suppose just suppose that this call has been done by call by reference. So, what has been passed here? The address of this and the address of this not the values, so no other copies have been made. So, now, I have got the local variable temp and here I do temp is assigned x. So, temp gets 10 then x assigned y. Now, what is my x and what is my y? The content of this address which have been passed will go to the content of x. So, this will be 5 and then y will be assigned temp. What will happen?.

This temp this value will go to the suppose I can make call by reference then this will go to this address, but if I just simply write in this way that does not mean it really does not tell me whether I am saying that I am copying the from the address of actually copying the value, but I am just taking telling a hypothetical case. So, this 10 will go there and this 5 would be changed. In that case when I come back from here and I print here then I would have got the change scenario reflected, but the mechanism writing simply like this

in C means it is called by value not called by reference. So, what is being passed is actually the value.

So, if I could, if I could do that then it was possible to have that interchange and there is a mechanism for passing the addresses and not the actual values by using a concept called I mean structure called pointers which are nothing, but addresses we will have to we will see that later if time permits. But in general remember that this interchange has not been possible by call by value in the way we wrote the program. So, with this we complete our discussion on parameter passing normally, but a distinction between parameter passing by call by value and call by reference.

(Refer Slide Time: 11:23)

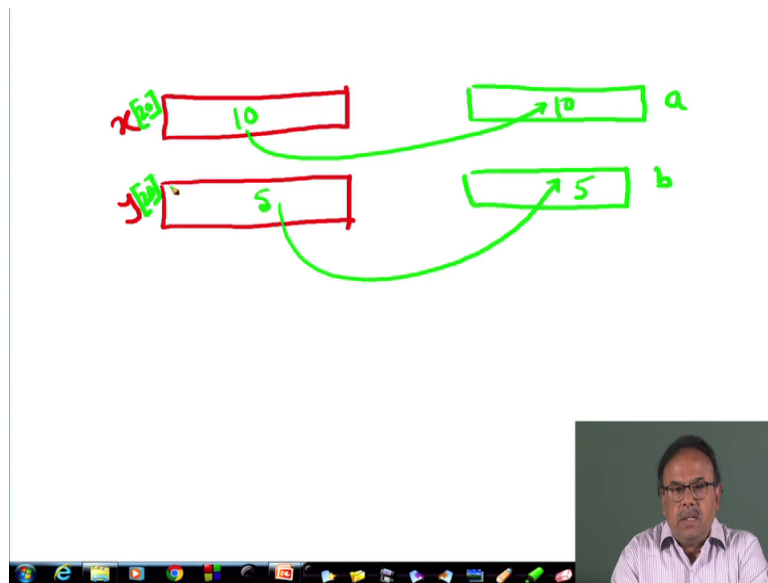
The slide is titled "Passing Arrays to Function". It contains a bulleted list of function calls:

- Array element can be passed to functions as ordinary arguments
  - `IsFactor (x[i], x[0])`
  - `sin (x[5])`

Hand-drawn in green on the slide is a diagram of an array. It consists of a horizontal rectangle divided into several cells. A small box is drawn at the end of the array, with a green arrow pointing from it to the `x[5]` argument in the `sin` function call. The number "2" is written next to the array diagram. At the bottom of the slide, there is a Windows taskbar and a small video inset of a man speaking.

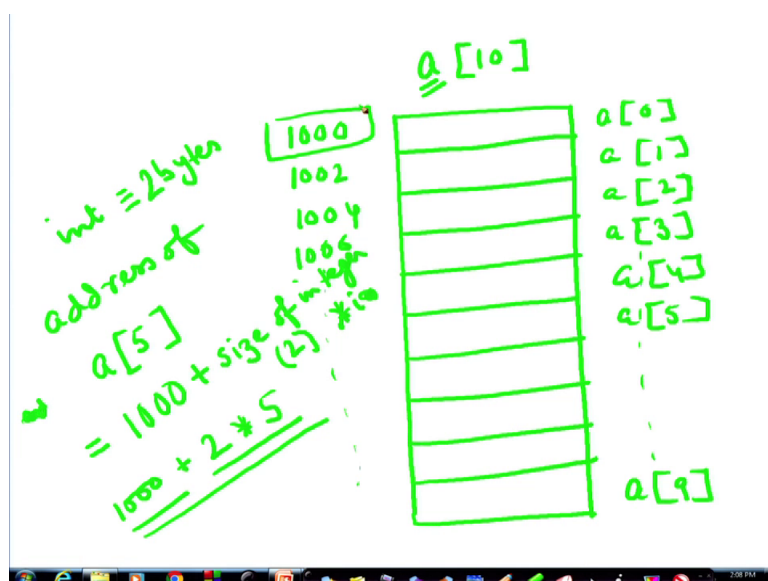
And we now start another very important point that is passing arrays to functions.

(Refer Slide Time: 11:45)



Now let us try to think. Here when we were having a variable x another variable y in the called function, in the caller function and I was having in my called function two other variables a and b, then the value of this would be copied here and the value of this was copied here. But suppose x is not an integer x is an array of 20 elements then all those 20 elements have to copy here and suppose this is an array of another 20 elements, so another 20 elements I have to copy here right. C allows only for arrays the parameter passing by difference. Now in order to understand that let us try to look at the structure of an array.

(Refer Slide Time: 12:56)

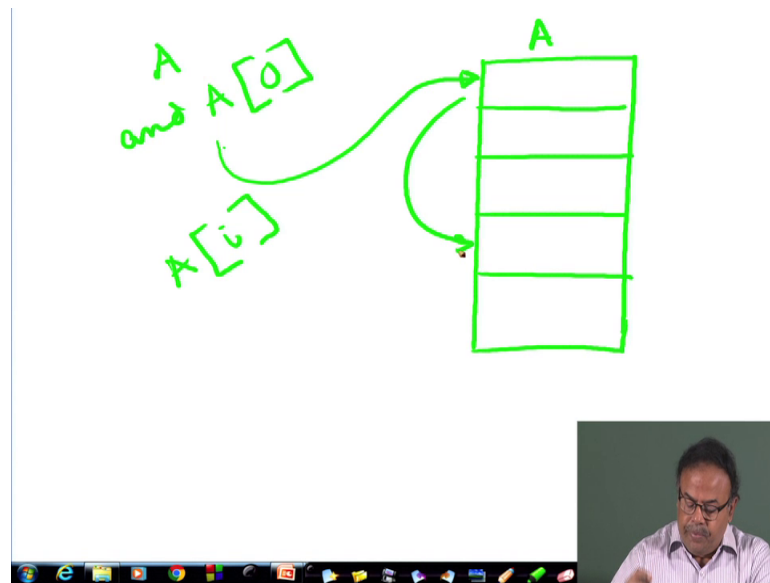


Suppose I have got an array  $a$  of 10 elements. I am not making a distinction between the size of the array, the actual size of the array, and the dimension of the array. I am assuming that the array  $a$  has got 10 elements. So, I have got  $a[0]$   $a[1]$   $a[2]$  up to  $a[9]$ , right these are the locations we know that. We also know that an array is allocated contiguous memory locations by the compiler.

So, all these are contiguous therefore, it is sufficient to know the address of the starting location, suppose this is 1000. If this is 1000 and if it be an integer and I assume that an integer takes 2 bytes say 16 bits and then this will be 1002 this will be 1004 etcetera I can compute any particular address, any particular address of this. So, if I say  $a[5]$  the address of this  $a[5]$  address of  $a[5]$  can be easily computed as the starting address thousand whatever is a starting address plus the index is 5; that means, and the size of integer which I know in a particular machine say 2 bytes times how much will be 106, 108. So, what will be 2 into whatever is a index minus 1; sorry  $a[5]$ ,  $a[5]$  will be the 6th element right. So, 5 times whatever this is suppose this  $i$ ,  $i$  times  $i$  right size of the integer times  $i$ .

So, it will be 1000 plus 2 times 5, 1010. So, this is  $a[0]$   $a[2]$   $a[0]$  will be 1006,  $a[4]$  will be 1008 and  $a[5]$  will be 1010. Since it is contiguous it is sufficient for me to know the starting address of the array, therefore, it is good enough to establish the correspondence between the name of the array and the starting point of the array. What I mean by that is again I draw this.

(Refer Slide Time: 16:46)



Now, I named the array differently A and suppose it has got 5 elements all right. Now A and A 0 are treated to be synonymous. When I say a; that means, I am actually referring to this element, this address, not this element this address. And since I know that therefore, A i depending on the value of i, I can compute where the actual location will be this is the fundamental concept we need to understand, all right. Therefore, we can pass the an array to a function as ordinary arguments.

For example, is factor whether x i is a factor of x 0, suppose I want to do that. So, you see is factor earlier I did x or y here I am writing x i, x 0. So, x i is what? Suppose x is an integer array. So, x i is an integer x 0 is another integer. So, I can simply pass this, sin of a particular angle. Where is that angle? In an array x an array x is there and in that the fifth, sixth element I am taking. So, this element is coming as the parameter. So, let us proceed a little further.

(Refer Slide Time: 18:38)

$A[0]$

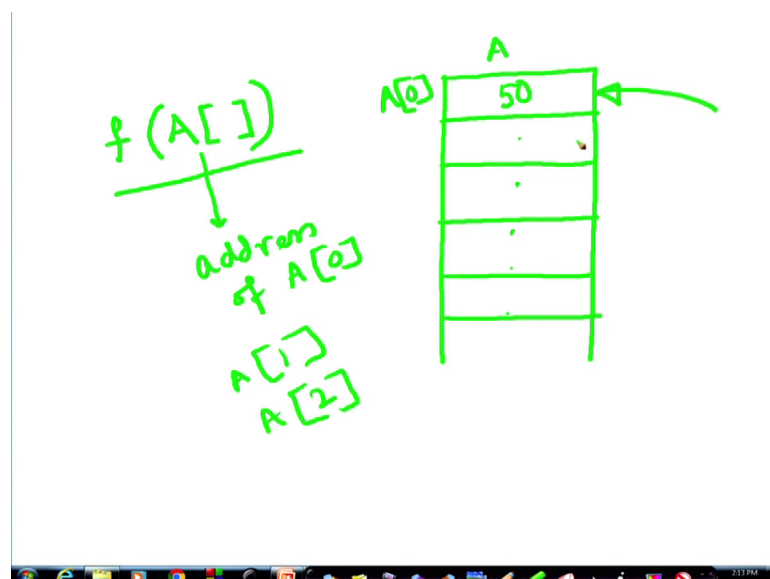
## Passing Entire Array to a Function

- An array name can be used as an argument to a function
  - Permits the entire array to be passed to the function
  - The way it is passed differs from that for ordinary variables
- Rules:
  - The array name must appear by itself as argument, without brackets or subscripts
  - The corresponding formal argument is written in the same manner
    - Declared by writing the array name with a pair of empty brackets

37

Now, that is for the individual elements  $x_i$  or  $x_{5 \times 0}$  I am just passing an element and if it is an integer array then an integer is being passed if it is a floating array, floating point array then a float will be passed, but what if I want to pass the entire array to a function. That is what I just now explained. That is an array name like  $A$  can be used as an argument to a function because  $A$  essentially means  $A_0$  and if I can pass  $A_0$  the address of  $A_0$  is known then all the elements are known because they are contiguous.

(Refer Slide Time: 19:46)





Now, the way it is passed differs from that of the ordinary variables. Why? Here when I have got an array and array A with A 0 when I am passing the array suppose somewhere in my function f I am passing the array A and just to say that it is an array I just do something like this. This is not the correct syntax, I will show you the syntax a little later.

Now, you see I write it in this way all right. Now that means, I am passing A 0, but if I pass the value of A 0 which might be a 50 I have got no clue about the other values I am not passing all the values. So, what do I have to pass? if I pass the address of A 0 address of A 0. So, this is what is being passed is address of A 0 then I can get access to all these elements A i, A 1, A 2 anything because I can compute the address very easily as I have shown just now, all right, I can do that.

So, this is an example this is a case where we call by reference we actually will pass the address and not the value of A 0. We are passing the address of A 0 and the address of A 0 is the same as the name A this is a very fundamental concept and let us try to understand this. How is it passed? The array name must appear by itself. So, now, we are talking about the some syntaxes, the array name must appear by itself as argument without brackets or subscripts the corresponding formal argument is written in the same manner. Let us look at an example.

(Refer Slide Time: 22:01)

### Whole Array as Parameters

```
const int ASIZE = 5;
float average (int B[ ])
{
    int i, total=0;
    for (i=0; i<ASIZE; i++)
        total = total + B[i];
    return ((float) total / (float) ASIZE);
}

void main ( ) {
    int x[ASIZE] ; float x_avg;
    x = {10, 20, 30, 40, 50};
    x_avg = average (x) ;
}
```

Only Array Name/address passed.  
[ ] mentioned to indicate that is an array.

(float) ~~total~~  
Type casted

Called only with actual array name

Say here, I am going to pass the whole array as a parameter. So, I have declared constant int a size 5. So, something some variable a size is assigned 5. Now, float average is a

function which is taking as parameter, sorry which is taking as parameter an array B all right, just the name of the B only array name or address of B is passed. Now, this symbol is mentioned to indicate that this is an array right. Now, let us see what is happening `int i total = 0, for i equals 0 i less than a size; that means, less than 5 i plus plus total equals total plus b i.` So, what is happening? So, here is an array B, here is an array A, a size the array size is 5 and all these elements are being added.

Now, all these elements are being added in this loop. So, B i only B 1, B 2, B 0, B 1, B 2, B 0, B 4 they are being added to total and then I am returning what am I returning I am returning total divided by a size now there is a new thing here also its better to look into that. Now, the array now average will be a floating point number and B is an integer array. So, sometimes, when i, so what has been done here is float a size when we do this float some variable x; that means, this variable is being type casted is being made to be represented as a float. So, if it was 5, 5 suppose then float A size. So, since is 5 float A size will make it 5.0.

(Refer Slide Time: 24:41)

### Whole Array as Parameters

```

const int ASIZE = 5;
float average (int B[ ])
{
    int i, total=0;
    for (i=0; i<ASIZE; i++)
        total = total + B[i];
    return ((float) total / ((float) ASIZE));
}

void main ( ) {
    int x[ASIZE]; float x_avg;
    x = {10, 20, 30, 40, 50};
    x_avg = average (x);
}

```

Only Array Name/address passed.  
[ ] mentioned to indicate that  
is an array.

$$\frac{120}{5.0} = 24.0$$

Called only with actual array name

And suppose the array was 1 2 0 4 5. So, 9 to 11, 12 total was 12. So, float total will make it 12.0. So, I am casting twelve forcing it to be represented as a float. So, then 12.0 is being divided by 5 with 5.0 and I am getting the average and that average is being returned here. So, that is that was not our main contention here that is the purpose of this

float. But the main contention here is that I have passed this array B, but who passed it the main function. Let us study the main function.

The main function x is an array of size A size that is 5 and x average is of some variable and x has been assigned as 10 20 30 40 50; x average I will compute, but I am calling this function average from here and I am passing x. What is x? x is an array and that is appearing just as a variable here and that is being accepted as B. So, what is a correspondence between x and B, x is somewhere suppose starting with location 5000 and there are 5 elements as I been said here 10, 20, 30, 40, 50.

(Refer Slide Time: 26:24)

### Whole Array as Parameters

```
const int ASIZE = 5;
float average (int B[])
{
    int i, total=0;
    for (i=0; i<ASIZE; i++)
        total = total + B[i];
    return ((float) total / (float) ASIZE);
}

void main () {
    int x[ASIZE]; float x_avg;
    x = {10, 20, 30, 40, 50};
    x_avg = average (x);
}
```

Only Array Name/address passed.  
[] mentioned to indicate that is an array.

x

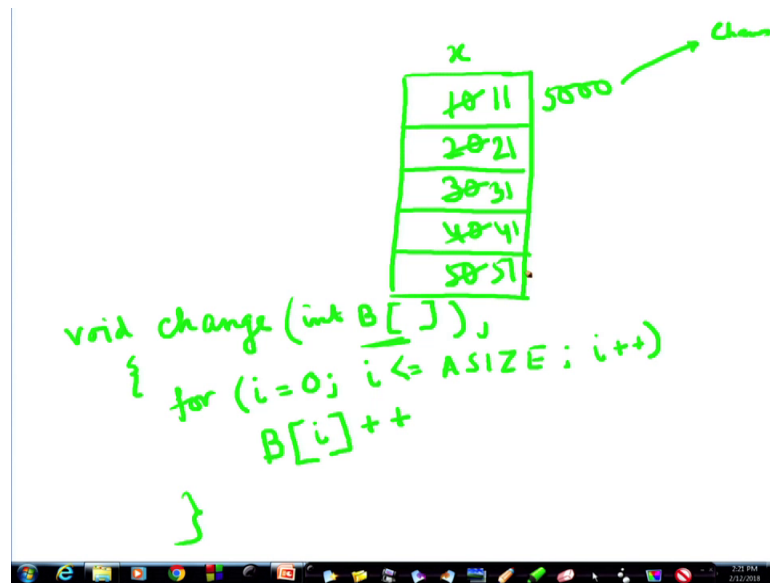
10
20
30
40
50

5000 → B

Called only with actual array name

Now, when this x is passed to B this function now knows that B is an array because it had this thing and this 5000 is passed to B. So, now, B knows where is B, B is the same array. Now, whatever change I do here, suppose in this function instead of computing the total if I had written this function in a different way for example let us do that and suppose I have got this function x, I have got this function x this array, sorry I am sorry x where 10 20 30 40 50 are there and in my function change, I can also say void change int B this and in the body of the function what I do is for please try to understand i assign 0 to this, i less than equal to A size or 5 whatever it is, i plus plus, B i plus plus say this. What will happen?

(Refer Slide Time: 27:29)



Inside this function `change` I will take the first one change it to 11, this one will be 21, this one will be 31, this only 41, this only 51 and when I return I made a change in `B`, but `B` and `x` are the same because this address 5000 was passed to the function `change` and this `b` got 5000 and so whatever change has been done here will be reflected in the main function when I return, clear.

So, that is the importance. So, here I have called it with the actual array name here of course, I computed a total and return some other value, but if I had just change it inside this function I could have made it void function and the change would have been automatically reflected in the name.


(Refer Slide Time: 29:47)

**Contd**

```
void main()
{
  int n;
  float list[100], avg;
  :
  avg = average (n, list);
  :
}
```

```
float average (int a, float x[])
{
  :
  sum = sum + x[i];
}
```

We don't need to write the array size. It works with arrays of any size.



39

So, here you see we do not need to write the array size it works with arrays of any size. For example, here void this etcetera where list is an array of 100 elements and average I am calling here float average int a is an integer, and float x is map to list all right, x is an array because here how is the correspondence done. This list, list is being mapped to this same address is being passed and n is being passed to a how many elements are there, all right. So, I need not specify the size because the size is already specified here. I have got 100 element array, this is my list, this is list right. So, I have got 100 element here I know. What I have passed to x is just the address. So, whatever it is 100 or 150 that be true for x also right. So, we do not need to write the size of the array.

(Refer Slide Time: 31:12)

### Arrays used as Output Parameters

```
void VectorSum (int a[ ], int b[ ], int vsum[ ], int length) {  
    int i;  
    for (i=0; i<length; i=i+1)  
        vsum[i] = a[i] + b[i];  
}  
void PrintVector (int a[ ], int length) {  
    int i;  
    for (i=0; i<length; i++) printf ("%d ", a[i]);  
}  
  
void main () {  
    int x[3] = {1,2,3}, y[3] = {4,5,6}, z[3];  
    VectorSum (x, y, z, 3);  
    PrintVector (z, 3);  
}
```

5	+	7	=	12
6		5		11

Now, similarly array is used as output parameters. So, suppose I am going to do vector sum; that means, I have got two arrays a and b. So, what I am doing here? I have got two arrays a, an integer array another integer array b and I am adding them. So, I will add this element say 5 with this 7 and I have got another array v sum, v sum where the sum will be stored so this will be 12, if it was 6 and this was 5 here we will store I will add these two and I will store 11 ok.

So, let us see how the vector sum. Now, vector sum will be void you know because it is doing the addition and the addition is remaining in this vector sum. So, I have got let us start with the main function x is an array. So, this is actually x a or let me say let me put the main function first. So, it will be x in the main function and a in the this function y slash b. I am writing slash b because these two was the same because the address is shared all right and vector sum will be z. So, z or vector sum. So, x is 1 2 3, y is 4 5 6 and z, z has not been initialized because, so let me let me; so that you are not confused, let me do it like this.

(Refer Slide Time: 33:28)

### Arrays used as Output Parameters

```
void VectorSum (int a [ ], int b [ ], int vsum [ ], int length) {
    int i;
    for (i=0; i<length; i=i+1)
        vsum[i] = a[i] + b[i];
}

void PrintVector (int a [ ], int length) {
    int i;
    for (i=0; i<length; i++) printf ("%d ", a[i]);
}

void main () {
    int x[3] = {1,2,3}, y[3] = {4,5,6}, z[3];
    VectorSum (x, y, z, 3);
    PrintVector (z, 3);
}
```

x/a	y/b	z/vsum
1	4	5
2	5	7
3	6	9

I have got an array that is x having values 1 2 3, another array y having values 4 5 6 and another array, another array z which will have the final value and z all right it is 0 elements, but we do not know the value. Now, I am calling vector sum and what am I passing x is being passed to this x goes to this. So, this is known as a for the function y goes to this. So, this is known as b. So, x and a locations are shared z and v sum the locations are shared, all right.

Now, I am calling vector sum. So, all these are being added in a loop 5 7 9, then I come out I come out. Now, you see when I come out of this function these are already reflected because it was passed by reference. Now, let us look at the print, print vector what is happening here. Void print vector that is another function that I am calling here z 3; that means, 3 elements will be printed from that z array z array was here which had that elements like 5, 7 and 9.

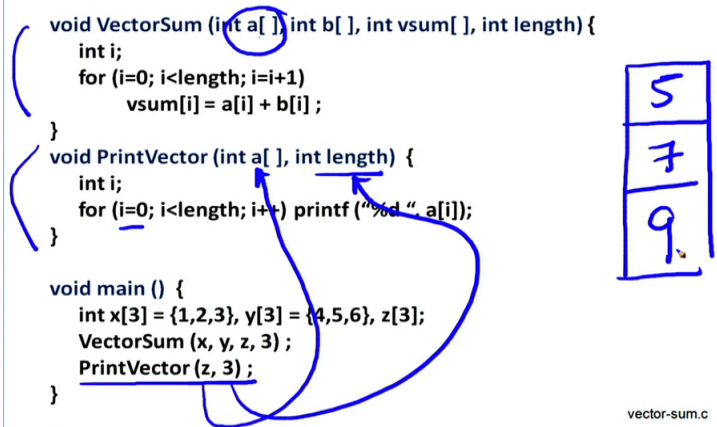
(Refer Slide Time: 35:29)

### Arrays used as Output Parameters

```
void VectorSum (int a[ ], int b[ ], int vsum[ ], int length) {
    int i;
    for (i=0; i<length; i=i+1)
        vsum[i] = a[i] + b[i];
}

void PrintVector (int a[ ], int length) {
    int i;
    for (i=0; i<length; i++) printf ("%d " a[i]);
}

void main () {
    int x[3] = {1,2,3}, y[3] = {4,5,6}, z[3];
    VectorSum (x, y, z, 3);
    PrintVector (z, 3);
}
```



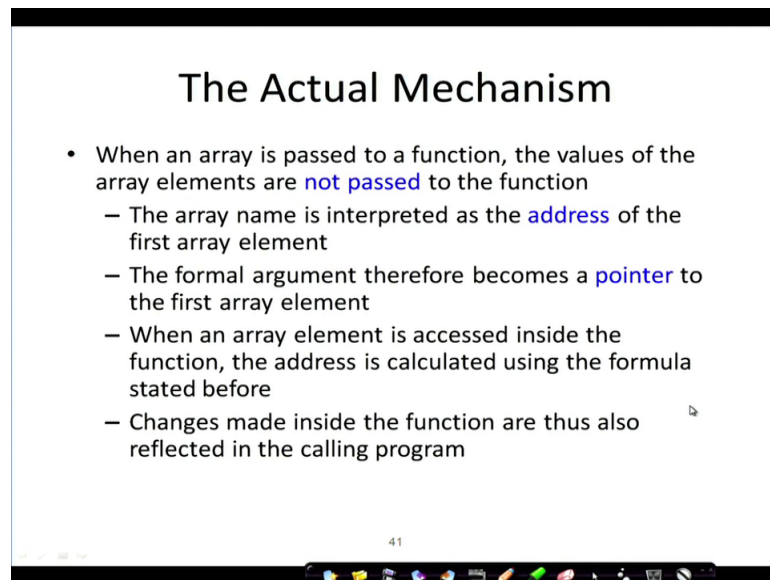
vector-sum.c

Now, with that print vector it is just in a loop  $i$  equal to 0,  $i$  less than length. What is length? Length is 3 and what is the array  $z$  is  $a$ . Now, this  $a$  and this  $a$  are different again, this  $a$  and this  $a$  are different. This  $a$ , this is this  $a$  was this functions  $a$  and this  $a$  is this function  $a$ . So, they are different. So, here they are printing 5, 7, 9. So, that is as an output parameter.

So, you see actually whenever I am reflecting on an array I need not pass it because essentially automatically passed right. So, that is what is very important. So, I hope you have understood this.



(Refer Slide Time: 36:29)



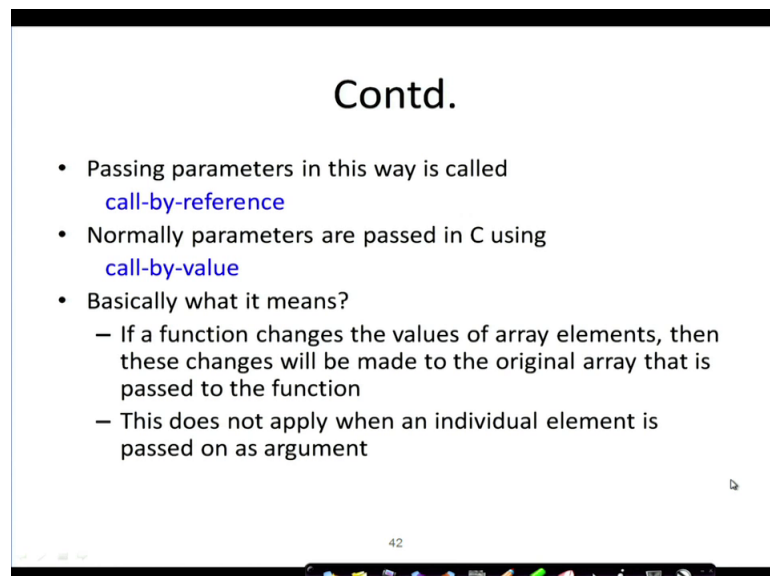
The slide is titled "The Actual Mechanism" and contains a bulleted list of points. The text is as follows:

- When an array is passed to a function, the values of the array elements are **not passed** to the function
  - The array name is interpreted as the **address** of the first array element
  - The formal argument therefore becomes a **pointer** to the first array element
  - When an array element is accessed inside the function, the address is calculated using the formula stated before
  - Changes made inside the function are thus also reflected in the calling program

At the bottom of the slide, the number "41" is visible, along with a taskbar containing various application icons.

The actual mechanism is when an array is passed it is the array elements are not passed, but what is passed is an address and the argument becomes a pointer to the first element or a pointer to the first element; that means, the address of the first element. And when an array element is accessed inside the function the address is calculated I have already explained the formula that is used for finding the array. So, this is known as call by reference.

(Refer Slide Time: 37:04)



The slide is titled "Contd." and contains a bulleted list of points. The text is as follows:

- Passing parameters in this way is called **call-by-reference**
- Normally parameters are passed in C using **call-by-value**
- Basically what it means?
  - If a function changes the values of array elements, then these changes will be made to the original array that is passed to the function
  - This does not apply when an individual element is passed on as argument

At the bottom of the slide, the number "42" is visible, along with a taskbar containing various application icons.

So, I think you have understood this. We will continue our discussions in the future lectures on some other important issues.