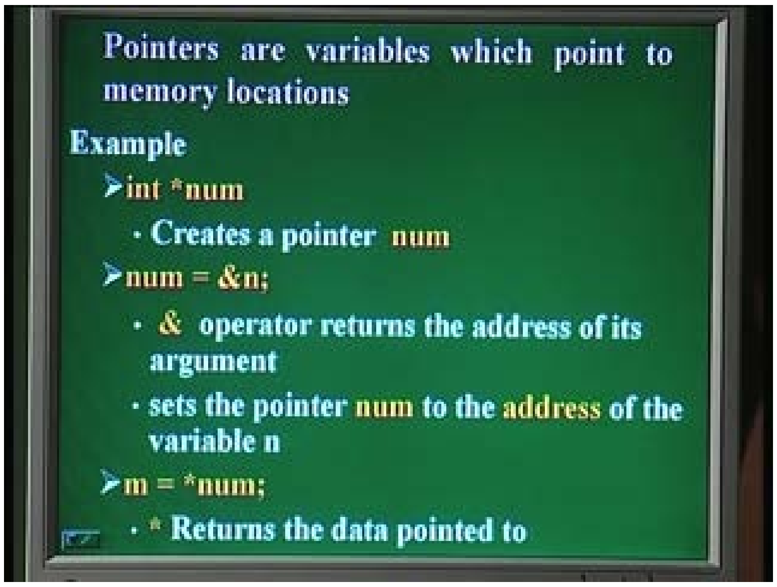
Numerical Methods and Programming
P. B. Sunil Kumar
Department of Physics
Indian Institute of Technology, Madras
Lecture - 3
Programing - Pointers and Arrays

Today's lecture on numerical methods and programing, we will continue our discussion on the pointers, the use of pointers. To start with, we will summarize what we did in the last class. So we looked at the representation of pointers in a program. So how do we actually put in a pointer, that is using the declaration of the pointer that is, here is an example. We used, defined a pointer called num, using this declaration. It has type integer and we use this asterisk operator to define this pointer. The pointer is defined as integer star num. So we, in the last lecture we actually saw the use of two operators; one is the asterisk, and the other one is the ampersand operator.

So we will see that again here okay. We created a pointer here called num, and now we are directing this num to the address of a number, n. So that is what the use of the ampersand operator is, basically to give the address of that thing. So that this operator returns the address of this argument n and then assigns that to num. So and then and this sets the pointer num to the address of the variable, n. That is what basically the idea is. The idea is to get the address of the operator n and the variable, n, and then give that to the pointer, num. Now we can, we will see the use of operator, asterisk operator. So that is now we are going to assign whatever that num pointing to that variable to another variable, m. That is what we see here, m equal to star m. So this asterisk returns the data to which it is pointing to.

(Refer Slide Time: 03:24)



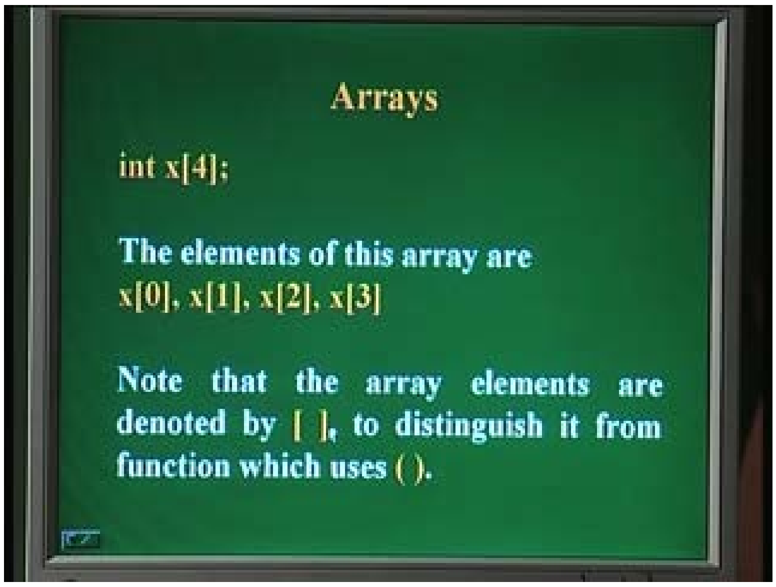
Pointers are variables which point to memory locations

Example

- int *num
  - Creates a pointer num
- num = &n;
  - & operator returns the address of its argument
  - sets the pointer num to the address of the variable n
- m = *num;
  - * Returns the data pointed to

That is one thing which we saw, and the next thing which we saw in the previous thing was arrays. So here is the declaration of an array. Arrays and pointers are the ones which we used interchangeably in the last lecture. So here is the declaration of an array again of type integer right. So array of dimension 4. For dimension, we are using this. The elements of the array would be now 0 to 3 because we dimension it as 4. The basic thing to note is that we use these square brackets instead of the brackets which we are used to when we write arrays in the nodes. This is because this bracket is used to represent functions.

(Refer Slide Time: 04:14)



Arrays

int x[4];

The elements of this array are
x[0], x[1], x[2], x[3]

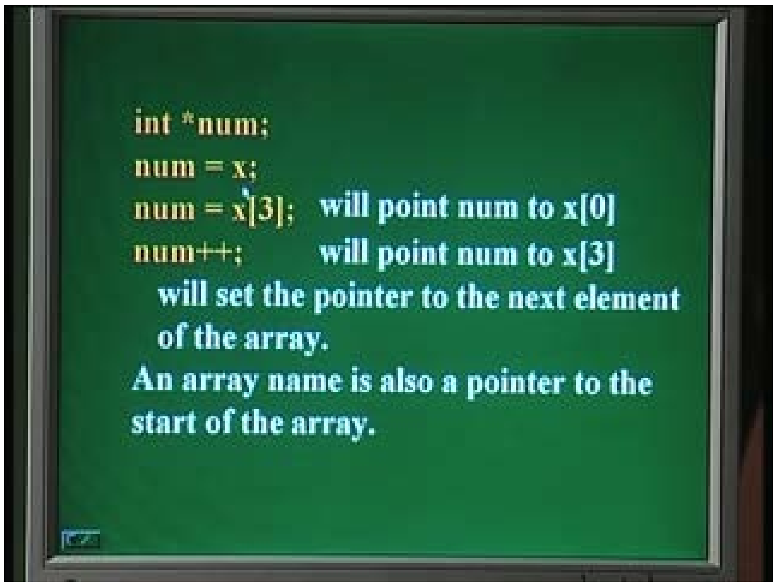
Note that the array elements are denoted by [ ], to distinguish it from function which uses ( ).

So to distinguish it from the functions, we use these square brackets. Something that we should keep in mind. So we will see now how we will use arrays and pointers interchangeably. So that is here. Now this thing tells you that. So we have integer, pointer, declared num okay, which is type integer okay. Now we say that num equal to x. x is an array. So if you say num equal to x, that will point to the base address of array x to the base element of x. That is what it will point to. So now if you say num equal to x of 3, then num will point to the last element, that is, the 3rd element of the array.

So we can increment if you set num equal to x, and if you say num plus plus, then it will point to the next element of the array. That is what we would see okay. This would point to x of 0. This would point to x of 3, etcetera. So now, if you do num plus plus, it will set the pointer to the next element of the array and we can read off all the elements in the array by doing this okay. Remember in this that x we said is an array, but x is also a pointer to the base address of that array. So x is an array and it is also a pointer to the base address of that array. So that is what we learnt from the last class.

Now how do we print out the address of each array element? So we can do that. So here is another array declaration. So this is also an example of declaring arrays. I have not only defined that array, we also assign numbers to it. I can do this this way also. So here is num is an array, and I have given it 10, 20, 30, 40, 50 as the elements of the array. So num of 0 is 10, num of 1 is 20, num of 2 is 30, num of 4 is 40 and num of 5 is 50. So I can define it this way. This is another way of doing it, and then now I have defined another pointer. So that is another integer-type pointer. We call it j okay.

(Refer Slide Time: 05:48)



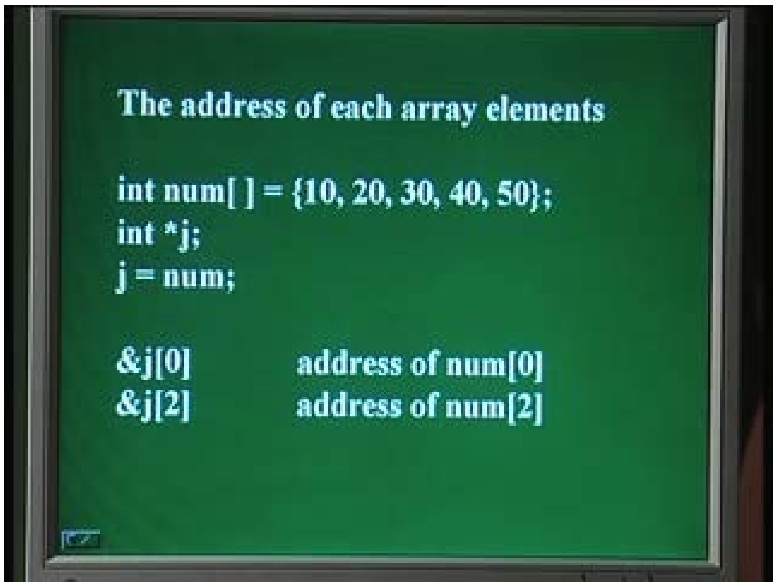
```
int *num;
num = x;
num = x[3];   will point num to x[0]
num++;        will point num to x[3]
```

will set the pointer to the next element of the array.

An array name is also a pointer to the start of the array.

So now this is the definition of a pointer. This is the definition of an array slightly different from the way we defined earlier, but this is another possible way of doing it. Now I assign j equal to num. This statement basically means that j would point to the base address of the array num. As I said, num is an array, but it is also a pointer to the base address. So j is a pointer, and now that is pointing to the base address of num. Now I can read off this as j of 0, as address of num of 0, and j of 2 as address of num of 2, or j of 1 as the address of num of 1, etcetera.

(Refer Slide Time: 07:26)



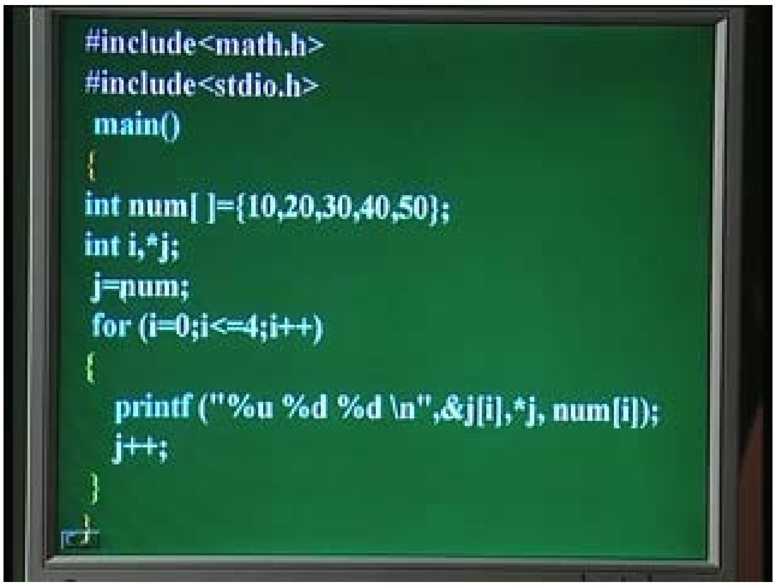
The address of each array elements

```
int num[ ] = {10, 20, 30, 40, 50};
int *j;
j = num;

&j[0]        address of num[0]
&j[2]        address of num[2]
```

That is the only way I can get the address of an array element. So you need to have a pointer which is pointing to the base address, and then you increment it. So that way you can get the, you can read off the address of each of the array elements, so we see this in a program. So here is a program which would implement this idea and prints out the addresses. So we have the usual “include” statements, and then I have declared this array, and it is of dimension 5. So it has num going from 0 to 4. Then I have this pointer, and also, I have this integer, i. This is the integer i and this is the integer pointer which is j okay. So I did j equal to num, and then I am going to print out now, by this statement, I can print out all the addresses of the array element.

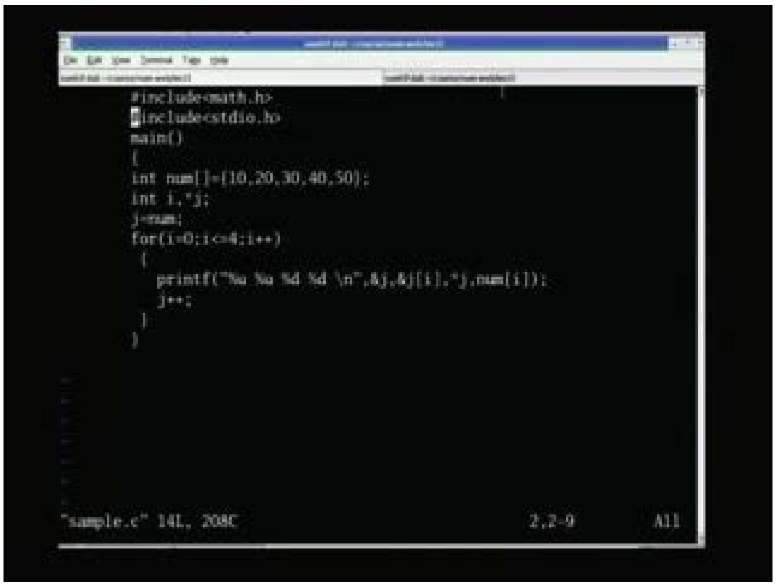
So here, percentage u is used to print out the array element. Percentage d is a syntax for printing out an integer, and again, this is an integer. So what will print out here is that it will print out the address of j of I, so i goes from 0 to 4. So it will print out the address of j of i, or the address to which j is pointing, j of i, and it will print out what value it is pointing to, and it will also print out num of i. That is what we can see. We will implement this program and run it. We will see that.

(Refer Slide Time: 09:03)



```
#include<math.h>
#include<stdio.h>
 main()
{
int num[ ]={10,20,30,40,50};
int i,*j;
 j=num;
 for (i=0;i<=4;i++)
{
   printf ("%u %d %d \n",&j[i],*j, num[i]);
   j++;
 }
}
```

(Refer Slide Time: 09:53)



```
#include<math.h>
#include<stdio.h>
main()
{
int num[]={10,20,30,40,50};
int i,*j;
j=num;
for(i=0;i<=4;i++)
 {
   printf("%u %u %d %d \n",&j,&j[i],*j,num[i]);
   j++;
 }
}

"sample.c" 14L, 208C                                  2,2-9         All
```

So here is the program. So here is what the program doing. Here again, I have declared the array. I have declared the pointer and the integer, and made the pointer to point to the base address of a num. Then I am incrementing i by steps of 1, from 0 to 4, and then I am printing out. I am printing out here the address of what j is, and then j of i is. So there is a subtle difference here. I am printing out j and the address to which j is pointing, and the address to which j of i is pointing to okay, and then the asterisk j which is the value of j pointer and num of i. So let us see what happens when you run this, so we compile that and we run it.

So we see what we are getting. So we are getting here the address, the base address j where it is pointing to. So that remains the constant. Even if I increment j plus plus, even if I increment j by j plus plus, the base address does not change. Remember the base address remains the same, but j of i is now pointing to different numbers, address of the different array elements. So j of i is pointing to different elements of the array. Then I am printing out asterisk j, and I am printing out num of i, and they are the same. That is what we expect it to.

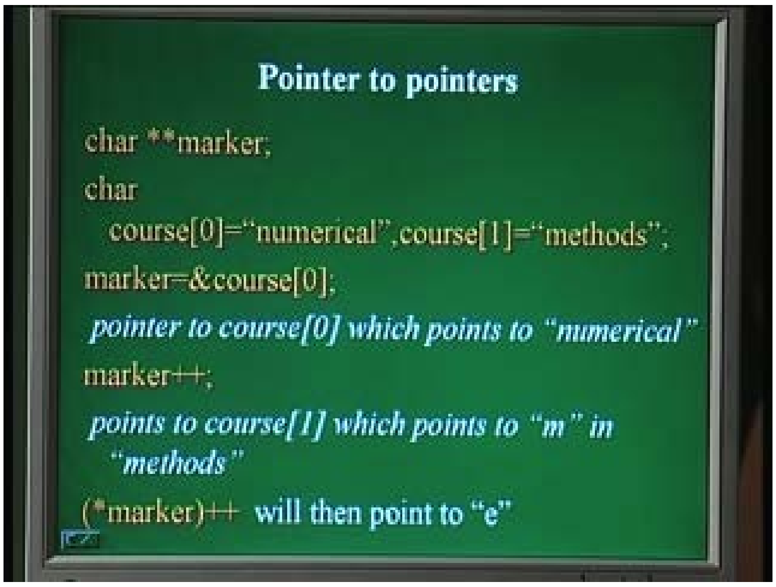
So there are two ways are getting an array element. I can point a pointer, integer pointer, to an array, and I can increment that integer pointer, and then pick up the value of that, or I could use the array elements straightaway as num of i, which we have used here, but if you want to get the address there is only one way of doing it. We have to say j of i. If you print ampersand of j, you will not get the address of each element. You will get only the address of the base, the base address. That is all we would get. That is a point to be kept in mind.

Okay so then we also saw, which I go through again, that we can make a pointer to point to another pointer. That is pointer to pointers. This will be useful especially when you want to read off character arrays and things like that, and also two-dimensional arrays. We will see that later. So here is an example. We have made a pointer here. So it is a pointer to pointer here. So it is double asterisk declaration. So star, star marker okay. Now then, I have a character array here. The array is called course 0 as numerical. That is one array; and then the first element of the character array is course 0, and that is given as numerical and then I have the second element which is course 1, and that will point to methods. Now you see here you have an array of pointers.

So that is what we are going to do. We have something called marker which is going to point to this. Let us see what happens. Okay so we say, we make the marker point to course 0, this element. So it will point to the base address of that. So that is what we have to do. That means pointer to course 0, which means, it points to numerical. That is the value given to that. The string is numerical, and that is what it is going to get and then we have, we increment the marker. So what will happen? It will go to course 1. That is what will happen. So it will point to course 1. That means it will be pointing to methods. Then you could ask the question, how do I read out each of the characters in the string? This is the way you can do it. You can say star marker plus plus. Before that nothing.

So we have this in the bracket here. That is important. We declare it as a pointer to pointer with a double asterisk. Use one asterisk, put it in a bracket, and then say plus plus. So when you say marker equal to course 0. It will point to numerical, and when you say marker plus plus it will point to methods. Having said that now, if you say star marker plus plus it will point to the next element of the array. Now we will use exactly the same when we want to work with two-dimensional arrays okay, so this is important. You remember this okay.

(Refer Slide Time: 13:57)



Pointer to pointers

char **marker;

char
  course[0]="numerical",course[1]="methods";

marker=&course[0];

pointer to course[0] which points to "numerical"

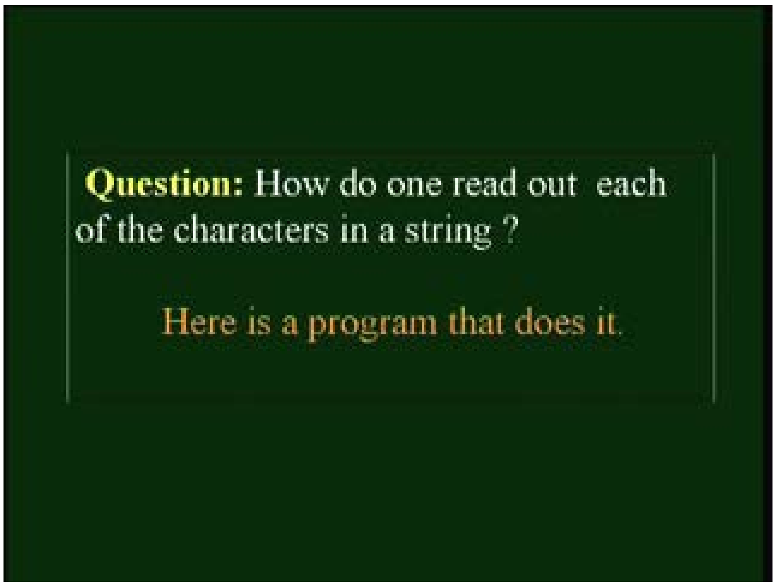
marker++;

points to course[1] which points to "m" in "methods"

(*marker)++ will then point to "e"

Okay so now we know how to declare pointers, how to declare arrays, how to use arrays, how to read out address of the arrays, and all those things. We know. So now we see how we can pass arrays to a function. We have to see something about the use of functions. We will see how to pass arrays to a function okay. So that is what we will see now. So now this is an important thing to remember, that in C all variables are passed by value, except arrays. Arrays are passed by reference.

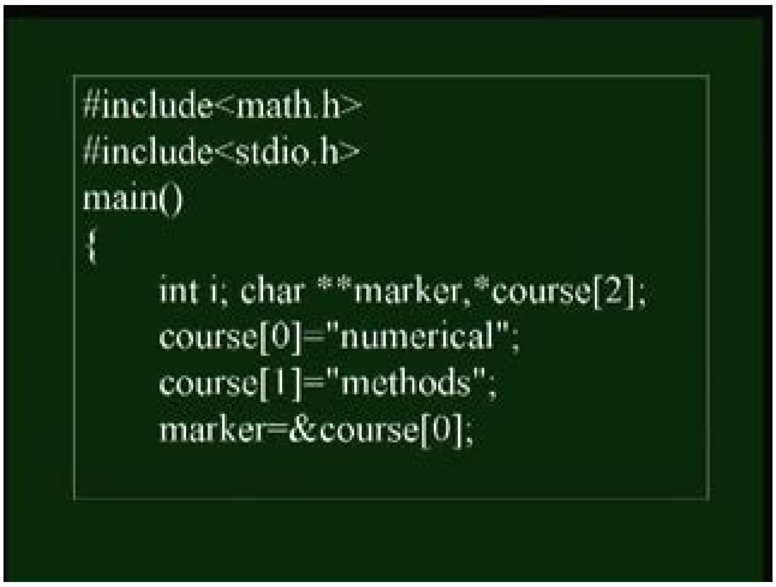
(Refer Slide Time: 14:04)



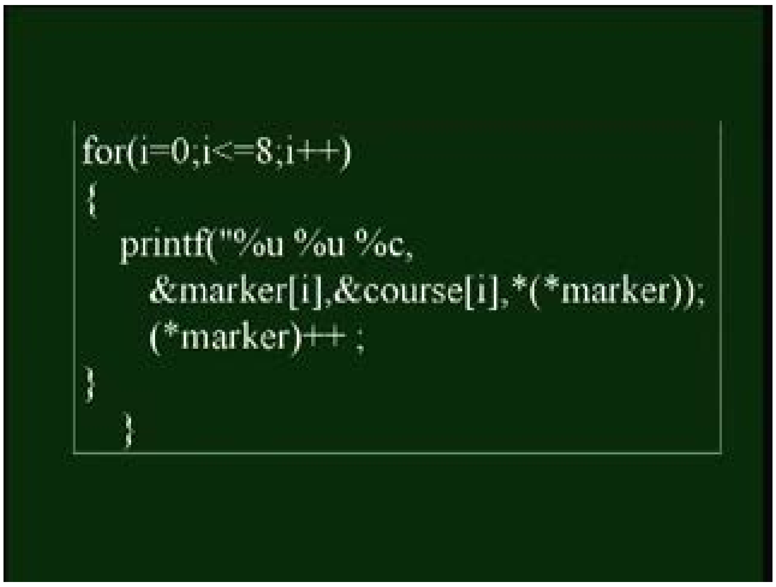
Question: How do one read out each of the characters in a string ?

Here is a program that does it.

(Refer Slide Time: 14:18)


```
#include<math.h>
#include<stdio.h>
main()
{
        int i; char **marker,*course[2];
        course[0]="numerical";
        course[1]="methods";
        marker=&course[0];
```

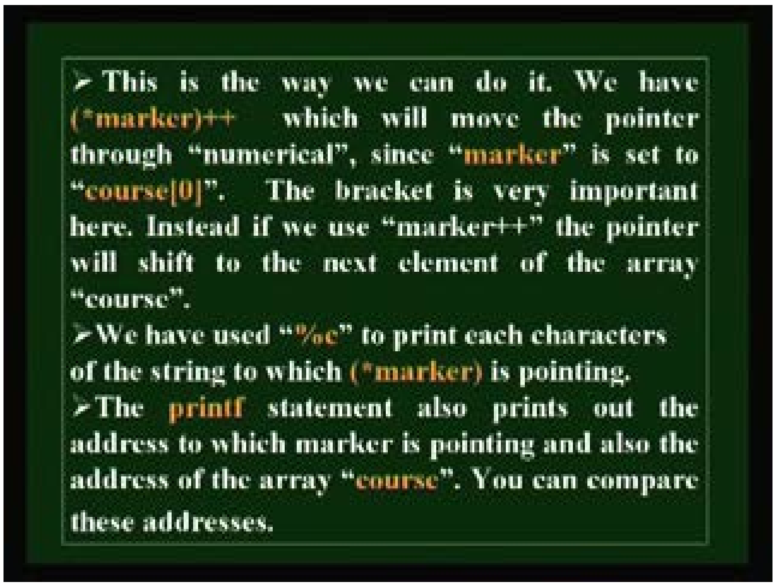
(Refer Slide Time: 14:33)


```
for(i=0;i<=8;i++)
{
   printf("%u %u %c,
     &marker[i],&course[i],*(*marker));
     (*marker)++ ;
}
   }
```

So now what do we mean by that, is that when you pass a variable to a function, the function makes a copy of that variable and works with that copy. That is because we pass only the value of that variable. Normally, when you pass a variable to a function, the function gets the copy of that variable, and it works with that variable okay. It does not, if you change that variable inside that function; it does not change that variable in that master program, master function, the main program. In the main program, you declare a variable, and you pass that to a function, and the function changes that variable. It will not change the value of that variable in the main program. So that is what we will see this with an example, when passed by value. As opposed to pass by reference, here we pass the address of the variable. So we have passed the address of the variable, which is some address of some memory location, and now in the function, if you change that value of the variable, it will change the value in the main program too.

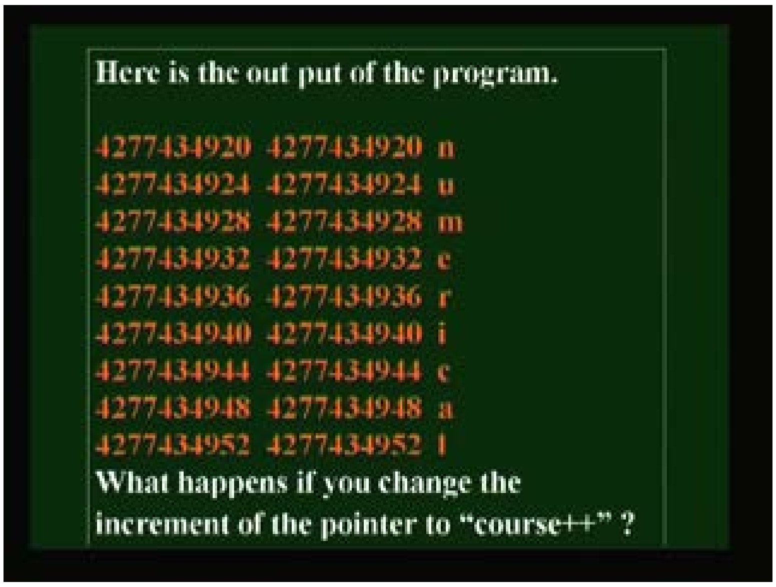
We are just working with addresses. We are not making a copy there. We are just passing the address, and we are working with that thing. So we are not making a copy here. If you pass by reference, it will not make a copy, but if you pass by variable it will make a copy. We will see that. When you are passing an array to a function, you are passing the pointer to the first element of the array. That is what I said. The name of the array is also a pointer to the base address. So when you pass that it will actually be pointing to the first element of the array.

(Refer Slide Time: 14:48)



- This is the way we can do it. We have (*marker)++ which will move the pointer through "numerical", since "marker" is set to "course[0]". The bracket is very important here. Instead if we use "marker++" the pointer will shift to the next element of the array "course".
- We have used "%c" to print each characters of the string to which (*marker) is pointing.
- The printf statement also prints out the address to which marker is pointing and also the address of the array "course". You can compare these addresses.

(Refer Slide Time: 15:04)



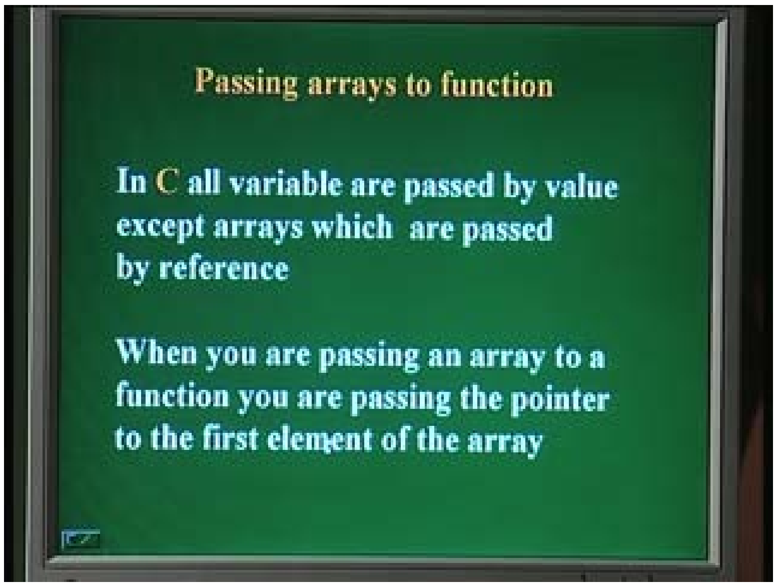
Here is the out put of the program.

```
4277434920  4277434920  n
4277434924  4277434924  u
4277434928  4277434928  m
4277434932  4277434932  e
4277434936  4277434936  r
4277434940  4277434940  i
4277434944  4277434944  c
4277434948  4277434948  a
4277434952  4277434952  l
```

What happens if you change the increment of the pointer to "course++" ?

When you work with that, it will change. If you pass an array to a function and if you change any value inside that function of the variable, if you change any of that, it will change the value of the array in the program from which it is passed. That is what we are going to see here.

(Refer Slide Time: 17:55)

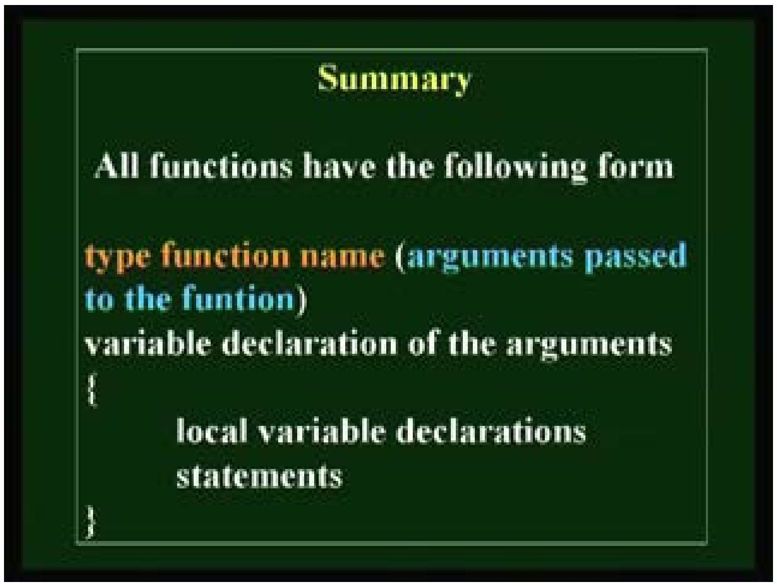


Passing arrays to function

In C all variable are passed by value except arrays which are passed by reference

When you are passing an array to a function you are passing the pointer to the first element of the array

(Refer Slide Time: 18:16)



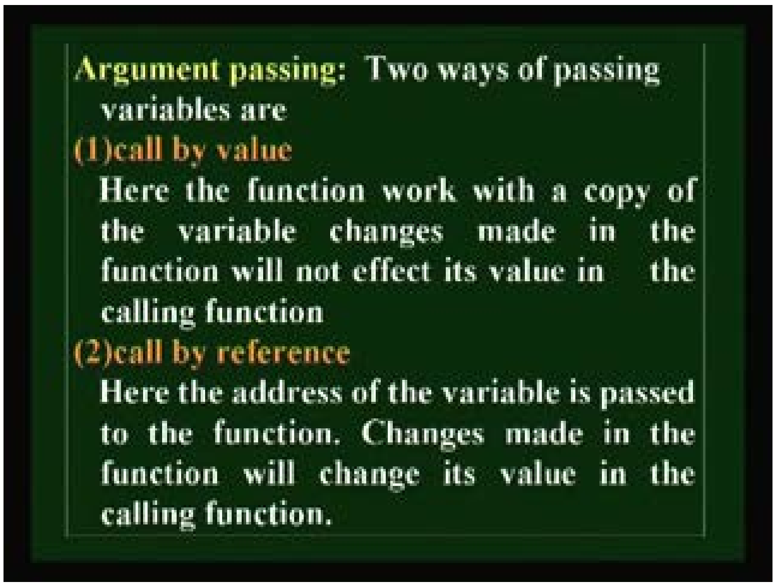
Summary

All functions have the following form

type function name (arguments passed to the funtion)
variable declaration of the arguments
{
local variable declarations
statements
}

So here is an example. We have an array declared, 10, 20, 30, 40, 50, num, and we have these integers here, which are defined as pointers. So we have 2 calls here. I will show you the program here. That is better. So, here we have a program. We have this declared as an array and we have integer i m and an integer point. Then we have an array dimension as m equal to 5. That is the dimension of the array. And then we point this pointer to the value of m. That is what we have done here. Then we are using 2 print statements. So we are saying, this is actually a function. Print is a function, and so is print 1, Print 1 is another function okay. So now, you see in print 1, I am going to pass the value of this function, 5, the number 5, and then I am passing the num. That is my array, and I pass j. Now let us look at the print, the print here okay.

(Refer Slide Time: 18:31)



Argument passing: Two ways of passing variables are

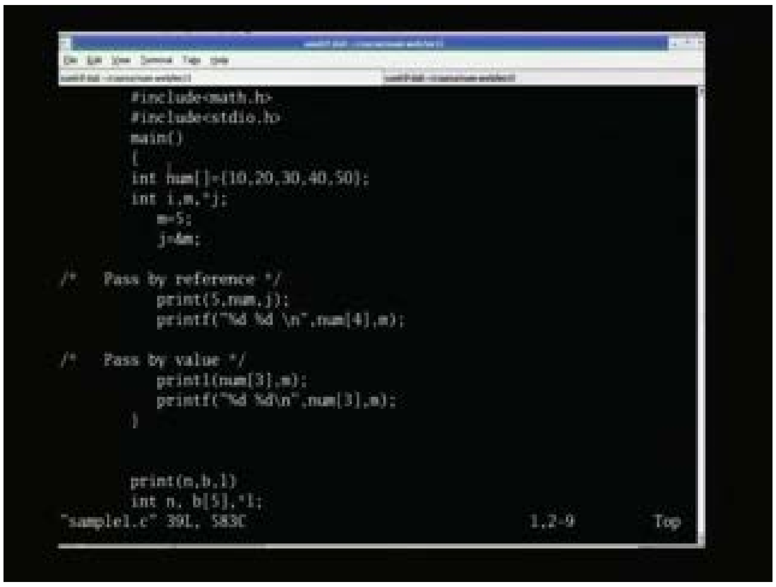
(1)call by value

Here the function work with a copy of the variable changes made in the function will not effect its value in the calling function

(2)call by reference

Here the address of the variable is passed to the function. Changes made in the function will change its value in the calling function.

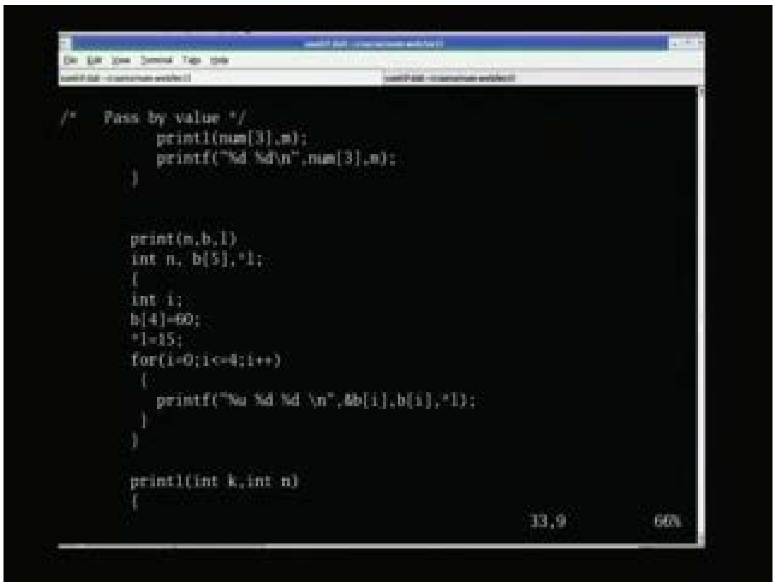
(Refer Slide Time: 19:10)



```
        #include<math.h>
        #include<stdio.h>
        main()
        {
        int num[]={10,20,30,40,50};
        int i,m,*j;
           m=5;
           j=&m;

/*   Pass by reference */
           print(5,num,j);
           printf("%d %d \n",num[4],m);

/*   Pass by value */
           print1(num[3],m);
           printf("%d %d\n",num[3],m);
        }


        print(n,b,l)
        int n, b[5],*l;
"sample1.c" 39L, 583C                          1,2-9         Top
```

Okay so that is the print function. The function takes n b, and I declare this separately. You can see that here. If you look at this function here, this is my function, and this is a call to the function okay. So I pass 5. That goes as n here, and I pass num. That goes as b. num is an array here, b is an array here declared. We have seen this before. Whatever is in here in the function you have to declare them here. So I have declared n as an integer, b as an array, and l as a pointer okay and I put here 5 num, and j, that is passed on to that function right. Okay so now then what do I do here? Okay I had here in this function, I had 10, 20, 30, 40, and 50. That is num of 0, num of 1, num of 2, num of 3, and num of 4 right.

(Refer Slide Time: 20:07)



```
/*   Pass by value */
          printl(num[3],m);
          printf("%d %d\n",num[3],m);
      }


      print(n,b,l)
      int n, b[5],*l;
      {
      int i;
      b[4]=60;
      *l=15;
      for(i=0;i<=4;i++)
       {
         printf("%u %d %d \n",&b[i],b[i],*l);
       }
      }

      printl(int k,int n)
      {
                                                  33,9          66%
```

The 4th element was 40 there, and I make that the 4th element of this into a 60 okay. That is what I do and then I also change this value. I assign this value as 50. I just assign that value as 50 okay I just assign this value as 50 and then I print out these things here and I also print out that here after the call. So two things I am doing. I will call, you know what the values of these array elements are okay. I call this function here, and then after calling the function I print out the values of those array elements here okay right and also the value m, and inside the function itself I print out the value of those functions, values of those variables. Both the address and the value I print out those here, and in the process I also change the last element here. Now you see what happens.

So we will run this and we will see what happens. We will comment out the rest of the program just to study only that part of it. So here we get the address. Remember, we are printing out the address of the array elements, and we are printing out the array element itself, and we are printing out this value of l. So that is what we are doing here, and that is what we get. So that is what we got, and so we got the address, and we got the number it is pointing to; you should remember that the last 1, instead of 50, now it becomes 60, because you printed out that as 60. Now we will go back here and whatever I am printing out here we are printing out after we call this. We are printing out the 4th element of num, the 4th element of num we are printing out here, and that is precisely what we changed inside this program. So num is mapped on to b.

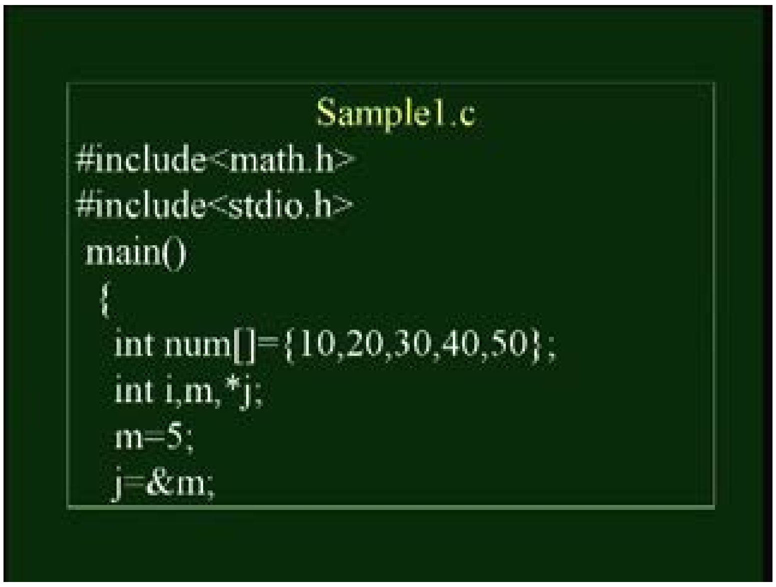
So when you said, when we call print like this, we are actually passing the base address of num to b. Then we took b of 4 and we made it into 60. Now we are trying to see, as num, even though we call num as b, here I changed b. Now the question is has num changed, and has m changed? That is what we are trying to figure out because m is j okay right, we again pass the address here of this integer. We pass the address, so what we see here is that both have changed. They become 60 and 50. That is what we see.

Remember, it was 5 and 5 and 50 it was right. So what was 50 here becomes 60 because I changed it in the function. And what was 5 becomes 50 because I changed it in the function, because I passed the address to the variables. Now let us see the second part.

Now we will comment about this part, and we look at another part of this program. Okay now pass by value. We saw the example of pass by reference. Now we look at the example of pass by value. So here again now, I call a different function, called print 1, which is given here. This function is called the print 1 okay. Now the print 1 is going to get everything by value. So we make a similar call here in the print 1, and I am going to pass only the 3 rd element of the array okay.

So I am going to pass the 3 rd element of the array and m. Remember, m is an integer right m is an integer okay and Num of 3 is the 3 rd element of the array. So instead of saying num now I am passing a particular element of that array, okay it is num of 3 and now in this function, I have defined all of them as integers. Now I am also using this example to show you two different ways of passing things into the function. You could have things return only the names of the variable written here, and declarations below that okay or you could have declaration itself in the bracket. That is also, I am using this to even demonstrate that. So here I have said integer k okay and integer n,

(Refer Slide Time: 27:48)

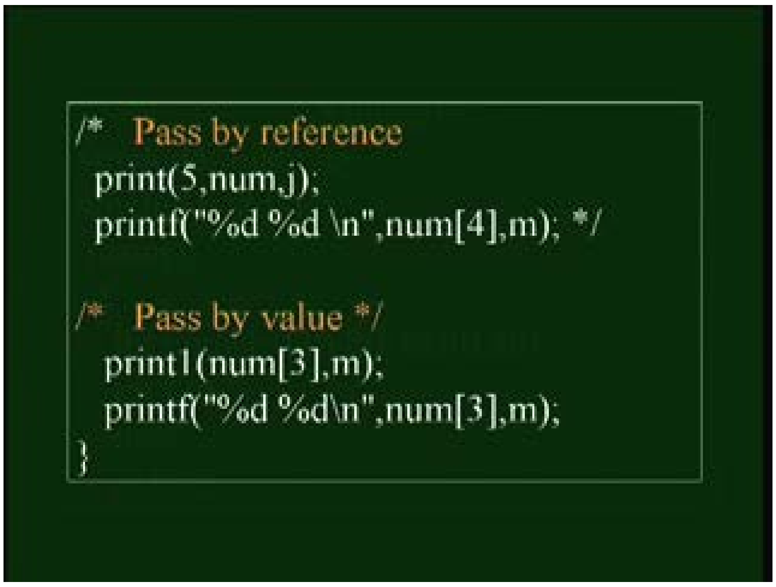

Sample1.c

```
#include<math.h>
#include<stdio.h>
 main()
  {
   int num[]={10,20,30,40,50};
   int i,m,*j;
   m=5;
   j=&m;
```

So there are two integers, and then I am passing them and then I change my k value and n value here. Remember this is the 3rd element, the 4th element of the array, and this is n and said so remember, the first argument was the 3rd element of the array, and the last argument was the value of m. Okay I print out here those values, and I print them out inside this function, and this is what we get. So what we should see is that if you do that, this value we changed it to 80 and 20, but inside the function it was 40 and 5 okay and it remains as 40 and 5 okay.

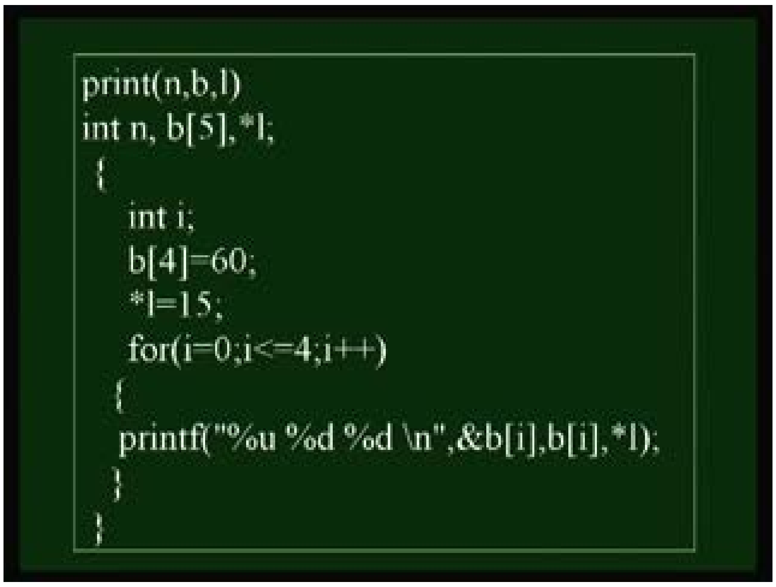
So you change the value of the variables inside the function, but that does not change in the main program. Okay the main program, whatever the value was, that remains as that. I said m is equal to 5 here, and num of 3 was 40, and that is what it is going to be. Then I print num of 3 and m, I will get 40 and 5, even though I change them inside this program, because here we have passed by value not by reference, I hope that thing is clear.

(Refer Slide Time: 28:02)



```
/*  Pass by reference
 print(5,num,j);
 printf("%d %d \n",num[4],m); */

/*  Pass by value */
  print1(num[3],m);
  printf("%d %d\n",num[3],m);
}
```

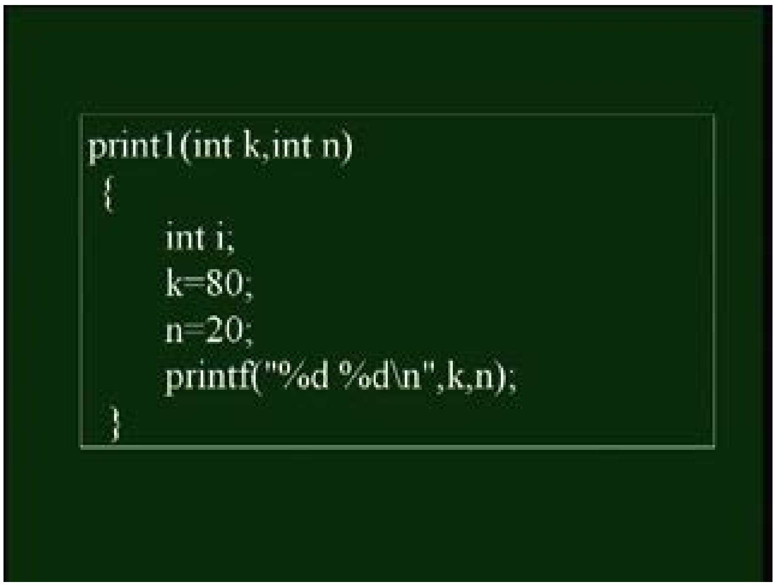
(Refer Slide Time: 28:17)



```
print(n,b,l)
int n, b[5],*l;
 {
    int i;
    b[4]=60;
    *l=15;
    for(i=0;i<=4;i++)
   {
   printf("%u %d %d \n",&b[i],b[i],*l);
   }
 }
```

So we will see the next thing is to go into two-dimensional arrays. So we will see. Okay so now, we dealt with one-dimensional arrays, pointers, and how to read off addresses, and things like that. Now we go into two-dimensional arrays and we will see how we will read off addresses here, and the values here. So here is an array which has 5 rows and 2 columns. Okay so this array is called arr, and it is 5, 2. This is not the only way to declare an array, but this is one way of declaring an array. I can also declare it as just 2 with an empty bracket in the first.

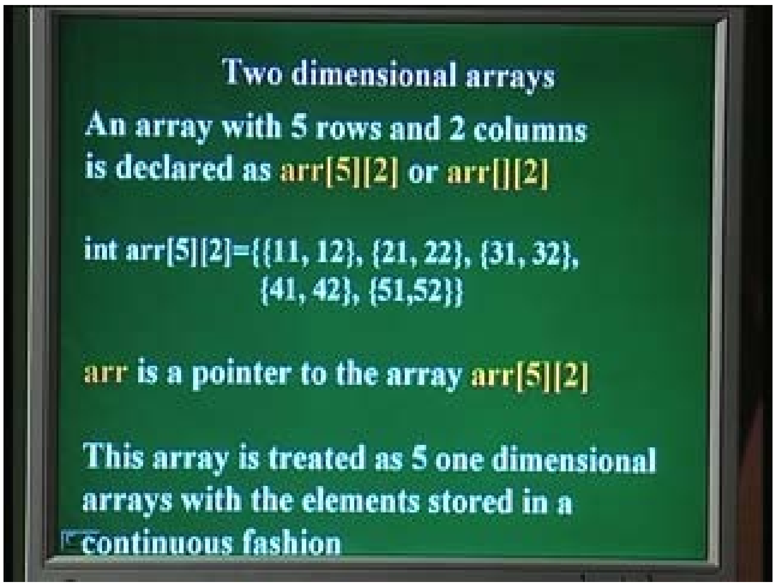
(Refer Slide Time: 28:33)



```
print1(int k,int n)
 {
       int i;
       k=80;
       n=20;
       printf("%d %d\n",k,n);
  }
```

So now the reason for this is, the array is treated as 5 one-dimensional arrays one after the other. So it is important that we declare correctly, this 2nd dimension. It is not important that this is declared, but this has to be declared because it is stored one after the other. So, if you do not declare this, then it gets messed up.

(Refer Slide Time: 30:58)



Two dimensional arrays

An array with 5 rows and 2 columns is declared as arr[5][2] or arr[][2]

```
int arr[5][2]={{11, 12}, {21, 22}, {31, 32},
               {41, 42}, {51,52}}
```

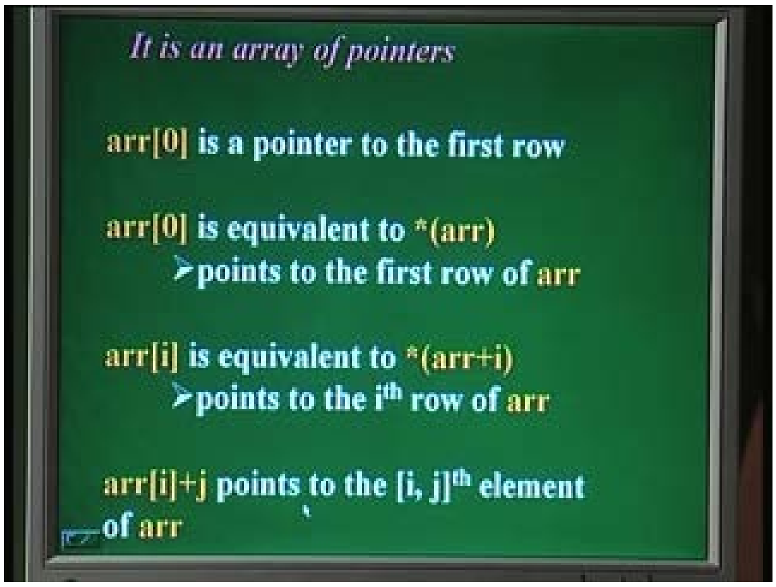
arr is a pointer to the array arr[5][2]

This array is treated as 5 one dimensional arrays with the elements stored in a continuous fashion

All the elements will get messed up. We can either declare it as arr 5, 2 or arr 2 okay. So here are the elements. I have declared all the elements. I have put in all elements here. So it is 11, 12, 21, 22, 31, 32, 41, 42, 51, 52 are the elements of this array okay. I have not filled up all the 5 rows, I have filled up 1, 2, 3, 4, 5, I have filled up all the 5. Now, if I say, as in the case of one-dimensional arrays, if I say arr, okay now what will it do? It will point to the base address of this array again, okay. That is same as in one-dimensional arrays okay. But the difference would be in actually reading out the elements of the array.

Okay so now we will see how to read out this array. First thing to remember that it is one-dimensional array is a pointer, and you remember that it is an array of pointers here okay. So, arr of 0 is the pointer to the 1st row of the two-dimensional array. So that is equivalent to saying that asterisk arr in bracket arr of 0 that is the base address okay, and then the first row of arr. Then, in general, if you say arr of i, it is equivalent to saying arr plus i. That is, it is pointing to the ith array, ith row of array arr.

(Refer Slide Time: 32:20)



It is an array of pointers

arr[0] is a pointer to the first row

arr[0] is equivalent to *(arr)

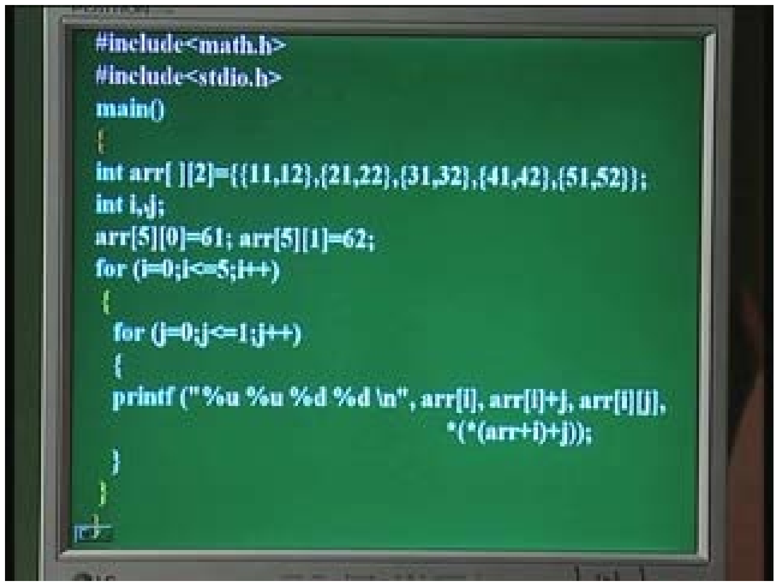
- ➢points to the first row of arr

arr[i] is equivalent to *(arr+i)

- ➢points to the i^{th} row of arr

arr[i]+j points to the $[i, j]^{th}$ element of arr

(Refer Slide Time: 32:27)



```
#include<math.h>
#include<stdio.h>
main()
{
int arr[ ][2]={{11,12},{21,22},{31,32},{41,42},{51,52}};
int i,j;
arr[5][0]=61; arr[5][1]=62;
for (i=0;i<=5;i++)
 {
  for (j=0;j<=1;j++)
  {
  printf ("%u %u %d %d \n", arr[i], arr[i]+j, arr[i][j],
                                   *(*(arr+i)+j));
  }
 }
}
```

So we had one-dimensional arrays, the name of the array as the pointer to the base address. Okay now here, we have an array of pointers. Here is the code which would implement this thing again. We will see the use of that in a program. If you want to pick up the jth element, we can say arr of i, i jth element, that points to the i th row, and then say plus j, then goes to the jth element. So that is the way we are going to point out. So we will see the use of that. Okay so here is the code which would, it is the same thing as that, so I just typed it in here. Okay so two-dimensional arrays, their declaration and reference that is what we are trying to demonstrate in this code okay. So we have declared the array, and we have 2integers, i and j, and now what we want to do is we want to change the array elements. This is the way of referring to the array elements, the 5th row and the 1st column. So I can change the array elements like this. So I have just added to that. So it is the 6th row. I have added one more row in to this.

So the 6th row and the 1st column, and the 6th row and the 2nd column right that is the correct thing to do. Sorry, it is 61 and 62. So I am just demonstrating the 2 ways of assigning the elements into the array. So I declare it as arr with an empty bracket, and then the 2 columns, and then I filled up 5 rows and 2 columns, and I am filling the 6th row by doing this. I could do this here or here. So it is just to explain, demonstrate to you, that you can do this in two different ways. Now I am going to basically print out the address of arr of i, and arr of i plus j, that is ij th element. That is, I go through the whole array. I go from 0 to 5 and 0 to 1.

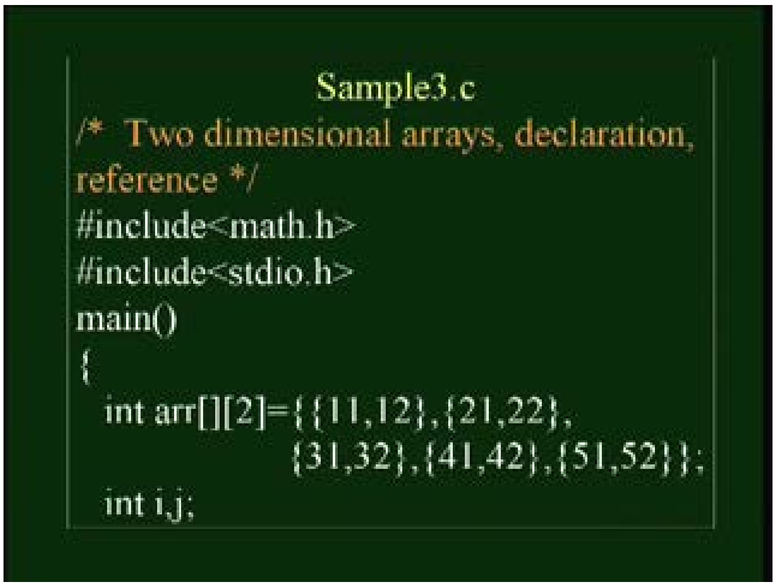
So I go through all the rows and all the columns. So that is what we are going to do. So we will run this, and then we will see clearly what that does. So that is what it is printing out. So what it is printing out. We come back here and we look at this thing. So this 1, 2, 3, 4 is what is printed here 1, 2, 3 and 4. We can see that. So we can see the last one. That is basically printing out all the elements in the array. So we see 11 12, and then, so first go through 1st row, and then print out the 2 columns. Remember, this is going through the rows and the columns.

So we take the 1st row and go to the 2 columns. That prints out 11 and 12, and then it changes the row to the next one and, prints out 21, 22, etcetera. That is what it is doing okay and I am printing out the address here. So printing out the first address is array arr i address. So now we can see that that is the address of the row. That is the address to the row. So if it is the 1st row, this is the address right. When we go to the next row, the address changes again. So within that row, when I change that next column that address does not change. The column address does not change.

So we can see how it is stored. We can see that this is the 1 st row, 1st column element. This is the 1st row, 2 nd column element. Then the row changes. So that is the 2 nd row, 1 st column, and then the 2nd row 2 second column okay and then it goes to the 3 rd row, 1st column, and so on, and you can see that it is stored in a continuous, in a serial fashion okay all the elements. We can see column by column it is continuous, sorry, row by row, in a row to column. That is why I said if you do not declare how many columns are there clearly then it will get confused. It will confused between whether this is the 1st element of the 2nd row, or the you know 3rd element of the 1st row. It would not know unless you actually tell what the number of columns are. That is why it is important to declare the second part.

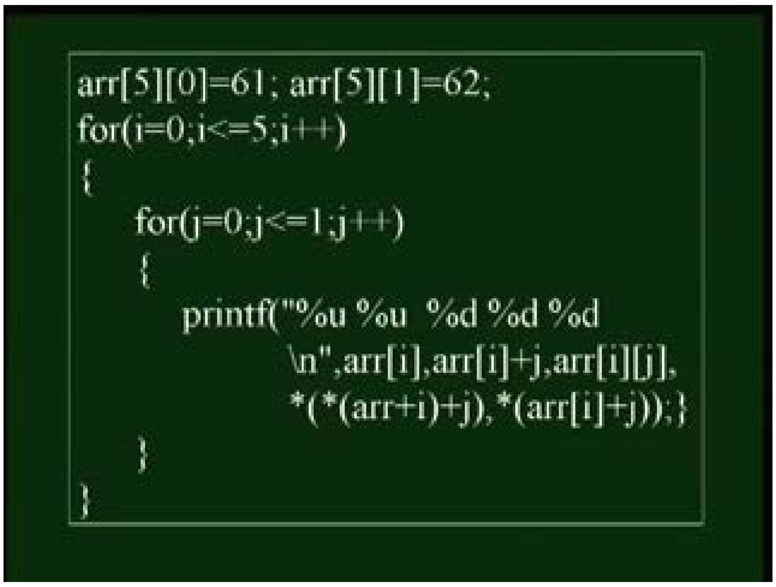
So now I have declared. I just wanted to show you here in this program that another way of actually printing out that number. Here I said asterisk arr plus I plus j, arr plus i in the bracket, and then plus j and then the whole thing in the bracket and another asterisk. You can get exactly the same number by doing asterisk arr plus i alone, so arr of I, that is an array. So we can pick up that array, and then plus j, and then do the asterisk of that because arr of i is also a pointer to that array. You remember that. That is what we see, and we get exactly the same number if you do this. That is 2 different ways of doing it. So that was the use of how do you read out the elements of an array of a two-dimensional array. Now we go to a pointer to this array.

(Refer slide time: 37:28)



```
Sample3.c
/* Two dimensional arrays, declaration,
reference */
#include<math.h>
#include<stdio.h>
main()
{
  int arr[][2]={{11,12},{21,22},
                  {31,32},{41,42},{51,52}};
  int i,j;
```

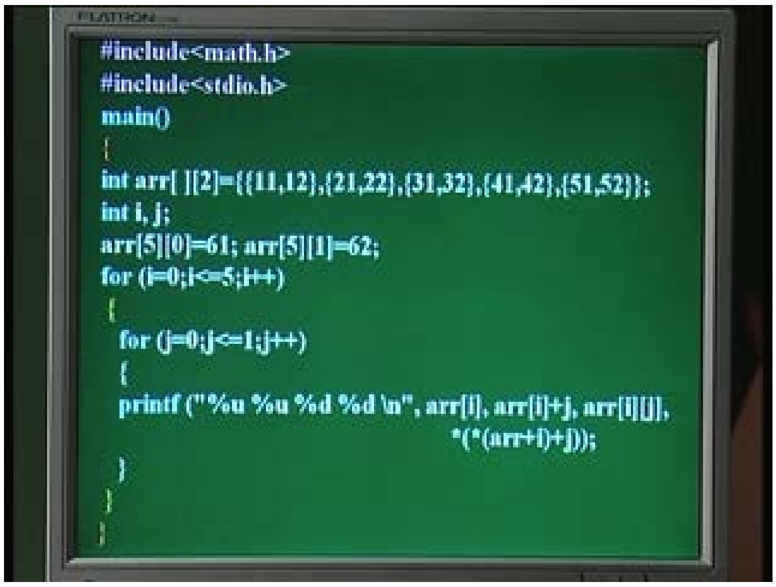
(Refer Slide Time: 37:43)



```
arr[5][0]=61; arr[5][1]=62;
for(i=0;i<=5;i++)
{
    for(j=0;j<=1;j++)
    {
        printf("%u %u  %d %d %d
                \n",arr[i],arr[i]+j,arr[i][j],
                *(*(arr+i)+j),*(arr[i]+j));}
    }
}
```

We go for two-dimensional arrays, and we said two-dimensional array is an array of pointers and I can read out each element of this array. I can get the address of each element of this array by using the arguments. Now we say that like we had for the one-dimensional array we had a pointer pointing to the one-dimensional array and we incremented the pointer and moving along the array and reading out the addresses we can do similar things here. Now here is a declaration. So here is an array, a two-dimensional array.

(Refer Slide Time: 38:44)



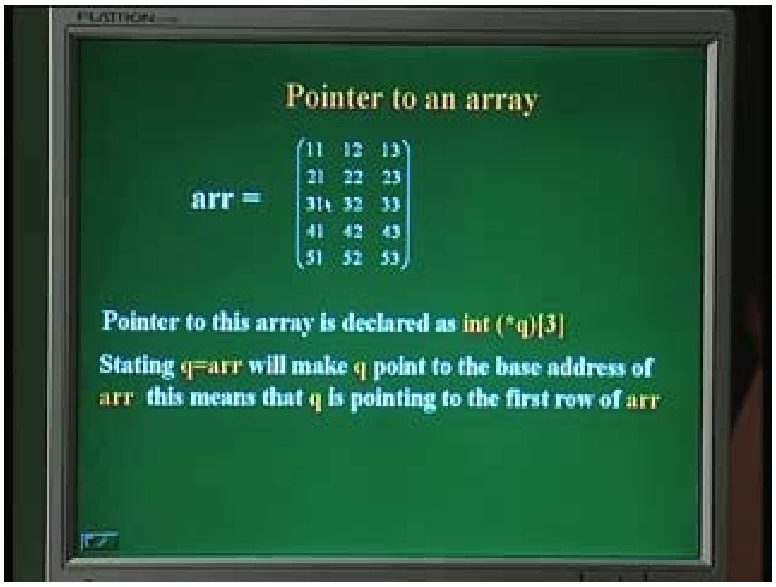
```
#include<math.h>
#include<stdio.h>
main()
{
int arr[ ][2]={{11,12},{21,22},{31,32},{41,42},{51,52}};
int i, j;
arr[5][0]=61; arr[5][1]=62;
for (i=0;i<=5;i++)
 {
  for (j=0;j<=1;j++)
  {
  printf ("%u %u %d %d \n", arr[i], arr[i]+j, arr[i][j],
                                        *(*(arr+i)+j));
  }
 }
}
```

Okay so here we see that we have different ways of printing out the address of two-dimensional arrays. So I showed you that here, that you can have for example, two different ways of printing out these array elements. I will just demonstrate that here again. I can use either an asterisk bracket arr plus i plus j or I can use asterisk of and a bracket arr of i plus j. These are two different ways of printing out the elements of the ij th element of a two-dimensional array okay. We can see similar things in the case of one-dimensional array too. So, for example, here in this statement it is a one-dimensional array. So the array is just num, as we have seen before, and I get two different ways of printing out that again. I could use a pointer and point on to that element and then I could print out that element, or I could print out b plus i because b here, remember this function here. I will go through this once again.

So here I am calling this function, to print, this print function, and this print function has got this array as b, num as b, and I am again using that here. So b is the base address to the array b, and I could say b plus I, and an asterisk to that asterisk bracket b plus i will again print out the same thing as asterisk j, so that is similar to saying here that I could use arr of i now, plus i, and then print out the address, print out the value. There are two different ways of printing out these numbers.

Okay so now we go into the next part. That is the pointer to an array and here is an array which is a two-dimensional array, so it has 1, 2, 3, 4, 5 columns and 3 rows, sorry, 5 rows and 3 columns, and now what do we read out the how do we put the pointer to this array? So here is, so we saw in the case that we have a pointer to this array by just saying j okay that is a pointer to an array. We could have an integer pointer which is pointing to this array. It is possible to do that. So we can have an integer pointer, which is just pointing to this array okay and similarly, we have an integer pointer which is pointing to this array, but this is a two-dimensional array. So we need an array of pointers to point to that. That is what it is. So q is an array of pointers okay and that is pointing to this. So stating q now, we just say q equal to arr, q would just point to the base address of array arr. That means, it is pointing to the 1st row 1st element. That is what we saw before.

(Refer Slide Time: 42:05)



Pointer to an array

$$arr = \begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \\ 51 & 52 & 53 \end{pmatrix}$$

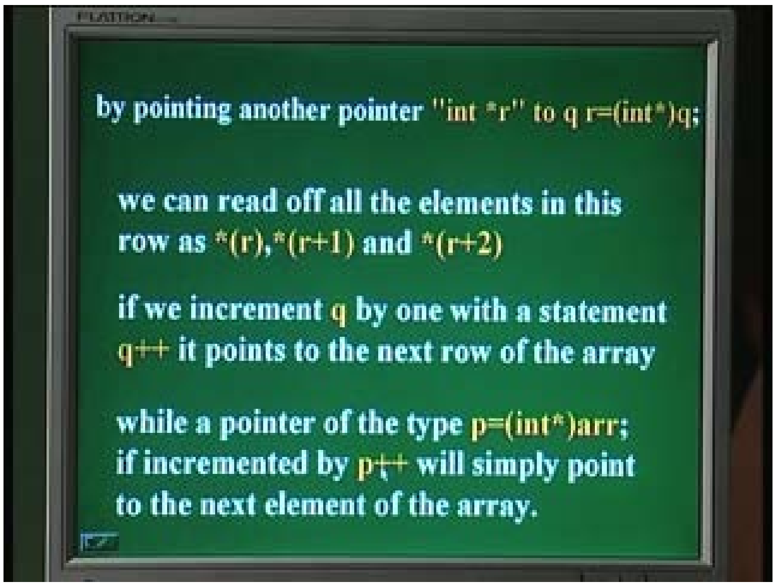
Pointer to this array is declared as int (*q)[3]

Stating q=arr will make q point to the base address of arr this means that q is pointing to the first row of arr

Now here is, how do we read out that? So now, once q, we have declared q as a pointer array, now we can declare another pointer which would point to that array q, and read off all the addresses from that, all the values from that. Okay so now here, for example, we have declared another pointer, r. Now that is going to point to q by saying r equal to you have to type it you have to typecast it. Now q is an integer pointer. r is another integer pointer. Then you, say r, you have to typecast it, integer, and then q that is the way you point a pointer to another pointer.

So we have to be careful with this. r is integer star q. So we will see that in a program, how do we use this. Then if you say asterisk r, asterisk r plus 1, asterisk r plus 2, that will give you the different elements of that row, and then you increment q, and then again you run through r. So that is what we would see. If we increment q by 1 with a statement q plus plus, it will go to the next row.

(Refer Slide Time: 44:06)



by pointing another pointer "int *r" to q r=(int*)q;

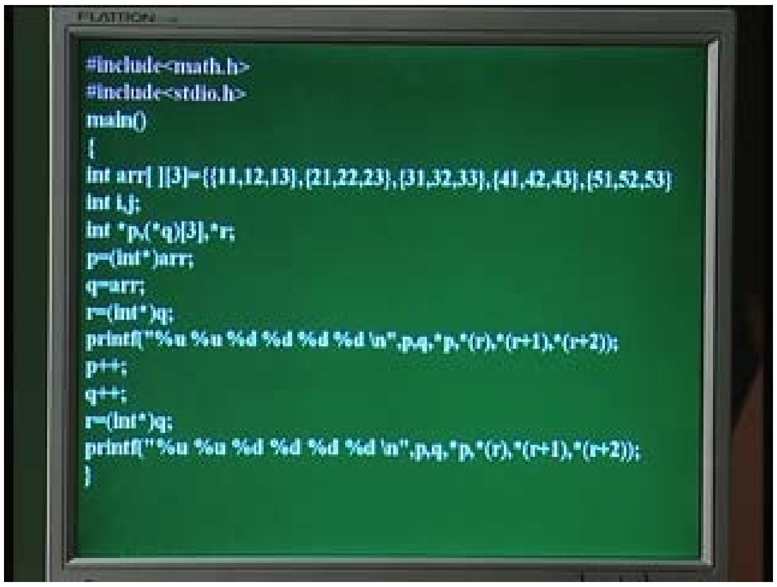
we can read off all the elements in this row as *(r),*(r+1) and *(r+2)

if we increment q by one with a statement q++ it points to the next row of the array

while a pointer of the type p=(int*)arr; if incremented by p++ will simply point to the next element of the array.

You go q plus plus, it will go to the next row of the array. So if you just declare one pointer p, not a pointer array, and then again you are going to point a pointer to a two-dimensional array, so you have to again typecast it. You have to say integer star array, and then if you say p plus plus it will simply point to the next element of the array. So that is the difference. If you say q is an array of pointers and that will point to the next row if you increment it by 1, while if you just say p s integer star array, and if you say p plus plus, it will go to the next element of the array.

(Refer Slide Time: 44:17)



```
#include<math.h>
#include<stdio.h>
main()
{
int arr[ ][3]={{11,12,13},{21,22,23},{31,32,33},{41,42,43},{51,52,53}}
int i,j;
int *p,(*q)[3],*r;
p=(int*)arr;
q=arr;
r=(int*)q;
printf("%u %u %d %d %d %d \n",p,q,*p,*(r),*(r+1),*(r+2));
p++;
q++;
r=(int*)q;
printf("%u %u %d %d %d %d \n",p,q,*p,*(r),*(r+1),*(r+2));
}
```

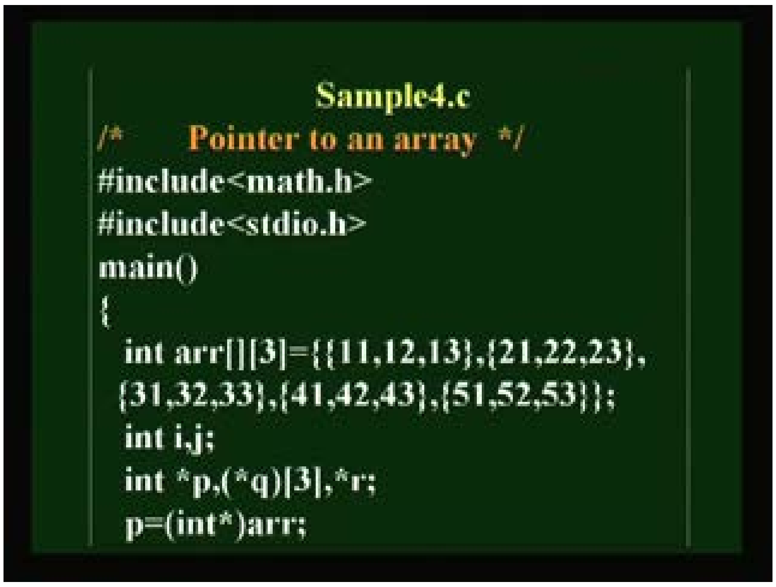
So there is no better way to understand this than actually looking through a program. So here is a program which would do this for us. So we will see that here too. Okay so here is the program. We have an array, an array which has 3 columns, and we have 1, 2, 3, 4, 5 rows. So we have 5 rows and 3 columns similar to what we have written there. So this is exactly the same as what we just saw okay and then I have 2 integers, integer i and integer j and I have 3 different pointers here. Two of them are just integer pointers and another is a pointer array. So we have 2 integer pointers and a pointer array.

Now we are making p equal to integer star arr. So what will happen? P would point to the base address of arr, and now I am making q point to the arr, so it will also point to the base address of arr. And then I am making r point to the first element of q. That’s 2 3 different assignment statements. Now we will print out, we will see what we are trying to do. We will print out the value of p, the address of p, the address of q, and the values to which p is pointing, the value to which r is pointing, and the value to which r plus 1 and r plus 2 are pointing.

So we will see that, and then we will p and q by 1, we will increment both, and again, once we incremented q, we again point r to the new incremented value of q, and again we will print out the same thing. Let us see what we get. So that is what we are going to see. Remember what we are trying to print out. We have p, q and r assigned this way, and now we are printing out, the first thing we are printing out is the address of p, so that is the same as the address of q, because both p and q are pointing to the base address of arr. So that is the same address. So you get the same address here. That is expected. You get the same address because both p and q are pointing to the same base address of arr. And then, what is the next one? The next one is the value to which p is pointing, and the value to which r is pointing. They are supposed to be the same because the first element of the array, the first element of the array is what these two are pointing to.

Now, that is the first row first column, and then we run through the column. We run through the row, the second column, and the third column. That is what we have. We have 11, 12, and 13..So we had 11, 12, 13, 21, 22, 23. So that is why we have ordered it. Now you can remember this as row in element, and this as the column, and this as the row. So the first row, first column, second row, first column, etcetera. So now, the next thing is we are looking at the value. We are incrementing p, and then we are incrementing q, and then we are again printing out the incremented values. So let us see. When p is incremented, the first one is we are printing out the p. When p is incremented, it went by 1 by 4 bits. It went to the next element of the array.

(Refer Slide Time: 49:32)

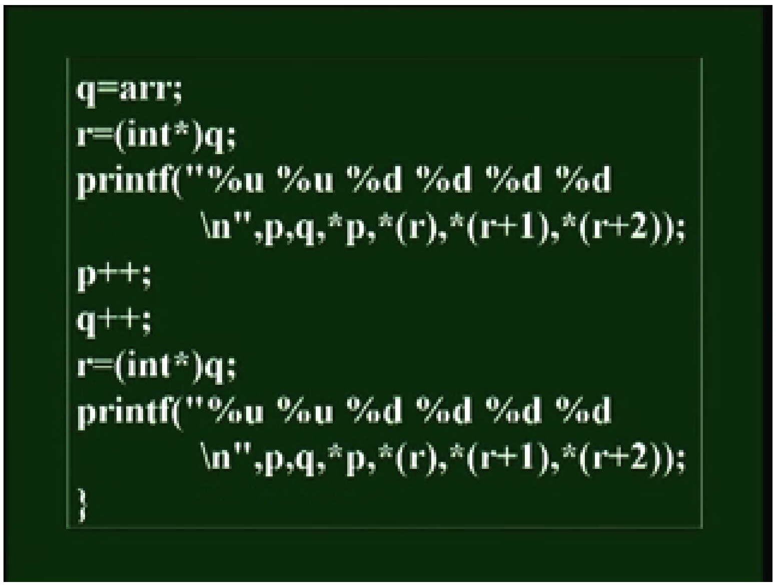


Sample4.c

```
/*      Pointer to an array  */
#include<math.h>
#include<stdio.h>
main()
{
  int arr[][3]={{11,12,13},{21,22,23},
  {31,32,33},{41,42,43},{51,52,53}};
  int i,j;
  int *p,(*q)[3],*r;
  p=(int*)arr;
```

So now you see, when we increment p and q they are not pointing to the same address. This jumps by 3. You can see that this jumps by 1. So when you have first assigned p and q to the base address of arr they have the same address, of course. We incremented p by 1 and q by 1, and we are printing out the addresses, and you can see they are not pointing out to the same address again, because p just simply went to the next element of the array, while q went to the next row of the array. So we have two different values. So now again we are printing out the r, r plus 1, r plus 2. So it reads out the next elements while p is simply reading the next element of the array right okay while r, which is pointing now to q, is reading the elements of the next row. So that' is the difference. So that is the difference between a pointer and a pointer array. these two things you see.

(Refer Slide Time: 49:46)



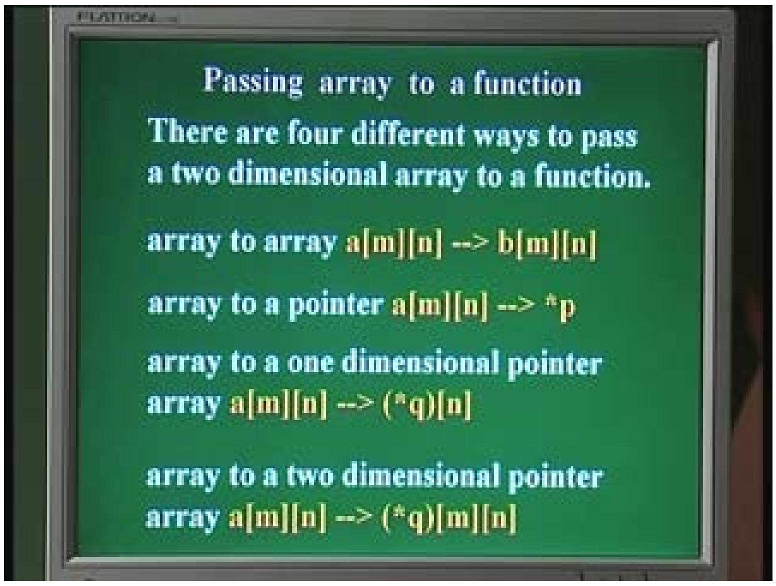
```
q=arr;
r=(int*)q;
printf("%u %u %d %d %d %d
        \n",p,q,*p,*(r),*(r+1),*(r+2));
p++;
q++;
r=(int*)q;
printf("%u %u %d %d %d %d
        \n",p,q,*p,*(r),*(r+1),*(r+2));
}
```

So next thing we should see is, now we know how to define two-dimensional arrays. We know how to define pointers and how to read out the elements of a two-dimensional array, etcetera. So now we see how we pass this array to a function. In many cases we need to pass an array to a function, and we have to see how that is to be done. There are 4 different ways of passing an array, a two-dimensional array, to a function. So now, passing an array to a function and returning an array from a function are slightly different. So here is the way of passing a two-dimensional array to a function. So how do we do that? There are 4 different ways, I said. So one is array to array.

We can pass an array in the call to the function. We can define an array, a, and in the function we can define another array b and then this would pass an array to an array. We will see in a program how this is done. Another way to do is to pass an array to a pointer. So this is the statement of p. An array, a, of dimension m by n, is passed on to a pointer, p simply. Another way of doing is an array to a one-dimensional pointer. We have seen that we can use a pointer, a simple pointer to point to an array, or we can use an array of pointers.

So now, we can use all that to pass the function, to pass it to a function. One is, you pass an array to an array or we pass an array to a pointer, or we pass an array to a pointer array, a one-dimensional pointer array okay or we could pass an array to a two-dimensional pointer array. So we have 4 different ways of doing this. So I think that is what we should now see in a sample program.

(Refer Slide Time: 51:53)


Passing array to a function

There are four different ways to pass a two dimensional array to a function.

array to array a[m][n] --> b[m][n]

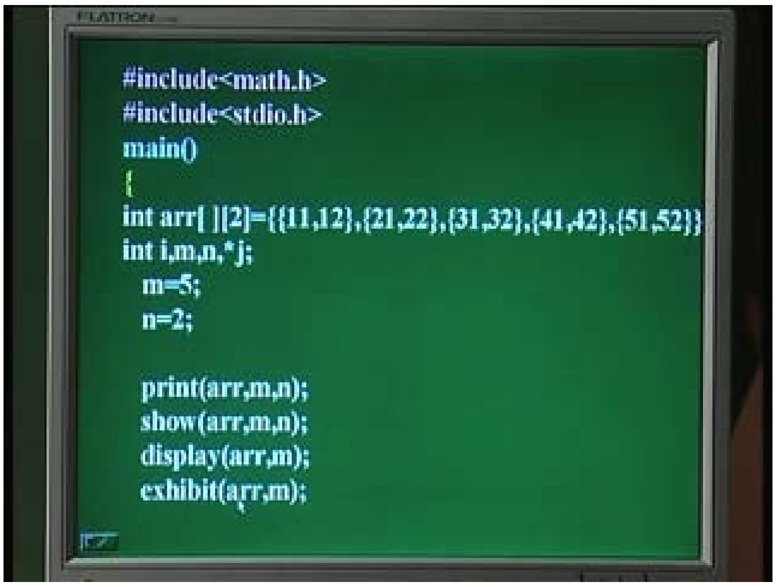
array to a pointer a[m][n] --> *p

array to a one dimensional pointer array a[m][n] --> (*q)[n]

array to a two dimensional pointer array a[m][n] --> (*q)[m][n]

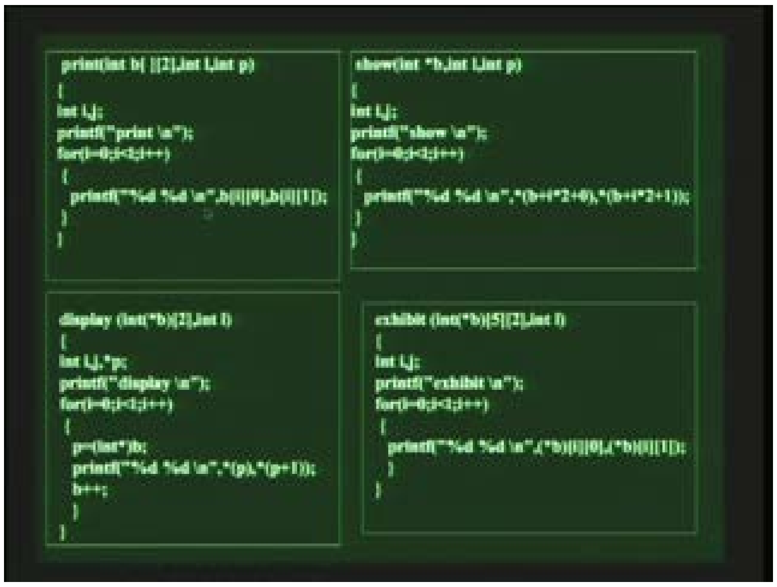
We will see the implementation of this in a sample code. So we have an array declared here which is a 5 by 2 array, 5 rows and 2 columns okay and now we have, I have declared 4 different functions. We will see an array to an array, an array to a pointer, an array to a one-dimensional pointer array, and an array to a two-dimensional pointer array. That is what we should see. We were talking about 4 different ways of passing an array to a function and these are the 4 functions which will demonstrate that.

(Refer Slide Time: 52:24)


```
#include<math.h>
#include<stdio.h>
main()
{
int arr[ ][2]={{11,12},{21,22},{31,32},{41,42},{51,52}}
int i,m,n,*j;
  m=5;
  n=2;

  print(arr,m,n);
  show(arr,m,n);
  display(arr,m);
  exhibit(arr,m);
```

So that is in the case of print dot c, so that is this function. So here, you will be passing an array to an array, and in this case you would be passing an array to a pointer array, and here in to an finally, into an array, into a two-dimensional pointer array. So an array to an array, and an array to a pointer array, and an array to a one-dimensional pointer array, and an array to a two-dimensional pointer array. That is 4 different ways of passing an array to a function.

(Refer Slide Time: 53:14)



```
print(int b[ ][2],int l,int p)
{
int i,j;
printf("print \n");
for(i=0;i<l;i++)
 {
  printf("%d %d \n",b[i][0],b[i][1]);
 }
}
```

```
show(int *b,int l,int p)
{
int i,j;
printf("show \n");
for(i=0;i<l;i++)
 {
  printf("%d %d \n",*(b+i*2+0),*(b+i*2+1));
 }
}
```

```
display (int(*b)[2],int l)
{
int i,j,*p;
printf("display \n");
for(i=0;i<l;i++)
 {
  p=(int*)b;
  printf("%d %d \n",*(p),*(p+1));
  b++;
  }
}
```

```
exhibit (int(*b)[5][2],int l)
 {
 int i,j;
 printf("exhibit \n");
 for(i=0;i<l;i++)
  {
   printf("%d %d \n",(*b)[i][0],(*b)[i][1]);
   }
 }
```

So that is what this particular program shows. So we will see that also in a program. That will be shown here. This program actually demonstrates those 4 different ways of passing an array to a function. So this is specifically to demonstrate that method of passing an array to a function. As I said, there are 4 different ways of doing it. So that is, array to array, array to pointer, and an array to a one-dimensional pointer array, and an array to a two-dimensional pointer array.

So all these functions are doing exactly the same thing. They just print out the values of these array elements in the respective functions. For example, print, here, is a function which receives that array arr as b array name here b it has got two columns, and what it does is it just simply prints out the array elements in the function but it receives this arr array as array name b. So remember, in this case any change in the values of the array made inside this function will also reflect in the main program because all these 4 ways of passing the array elements are all by actually by reference, not by value.

So any change you make here will also be reflected in the main program, but here we are not making any change right now, just printing out those values. That is what the print function is doing it is just receiving the array as an array. Now let us see what the show function is doing. In the show case, that array arr remember, that was the first element of the function call that is here, is received as a pointer. So integer star b is the one which receives the array elements. So now, how do we get the access to the array elements? So then it goes this way. So we go from i going from 0 to l. Now l is the dimension of the number of columns of the array which we had passed as 2. So that means 0 and 1.

So you will go here and use the star b plus i star 2 plus 0 and b plus i star 2 plus 1. These are the 2 elements of 1 row, and then we do that for all columns. So let us see that again, the call function. So we are calling here print, sorry, show, this function called show, the show to which we are passing this two-dimensional array which we define here, and we also pass the number of rows and the number of columns, so that is m and n and the function, show, actually receives that as an integer pointer b, and the number of rows are received as l and the number of columns as p.

So now this is just one pointer b, which is going to read out the entire elements of the array and the array elements are stored serially. That is, you go along the, go in a row along the columns, and then come back to the next row, etcetera. So that is what this is doing. So it is printing out each of the rows' elements. The first element of the row first, when i equal to 0, it is printing the first element of the first row, it as b plus 0 and b plus 1 the star here because this is a pointer.

You are printing out the value of what it is pointing to, and then we increment l. Then we will go to the next row and print out that, etcetera. So now let us see what is the next thing. That is the function called display, and now display takes array, passes array to a one-dimensional pointer array. So now we are just passing the number of rows into this one-dimensional pointer array, and this function display receives it as a one-dimensional pointer array. So each of those pointer arrays have 2 elements. That is what star b, this asterisk b, 2 elements in each of those arrays. So then to print out now the values, we have to use the pointer. We have to use another pointer which is now here p.

Now this pointer p has to be pointed to each of these rows, and then you have to move along the column to read out those elements. That is what it is doing here. [58:04] So p is now pointing to the base address of this array, one-dimensional pointer array b, p is pointing to b, and then p and p plus 1 will give the 2 elements of the row okay and then we go b plus plus. That means we will go to the next row of this pointer array, and then we again use p to point to that, and read p and p plus 1. So if you want, if you pass an array into a one-dimensional pointer array like this then you need to use another pointer to go through those array elements and read out them. The last method is to point, to pass an array to a two-dimensional pointer array, and that is shown in the last program. So the last function here, that is equal exhibit.

Okay and in this function, we receive it as a two-dimensional pointer array of dimension 5 by 2 and then now to print out those elements which are similar to printing out the elements of a two-dimensional array, but only thing, we have an asterisk sign here because this is a pointer. These are the four different ways of passing an array into a sub function. We will stop today's lecture at this point.