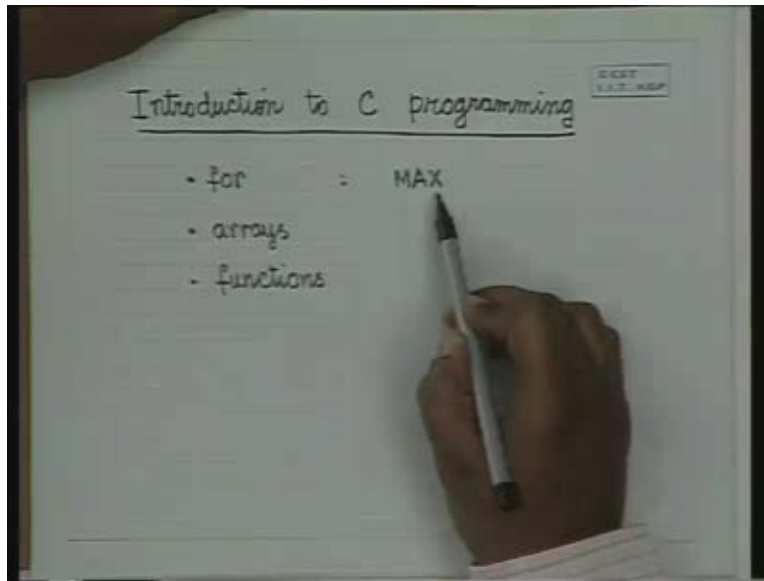


**Programming and Data Structure**  
**Dr. P.P.Chakraborty**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture # 03**  
**C Programming - II**

We shall continue our study of C programming language and in the previous lecture, we have just seen how to write simple programs to print variables, read variables, write the assignment statements and we have also seen the if then else construct and we shall next see the for loop, the looping statement and for this we will solve the example which I took in the first class finding out the maximum of n numbers.

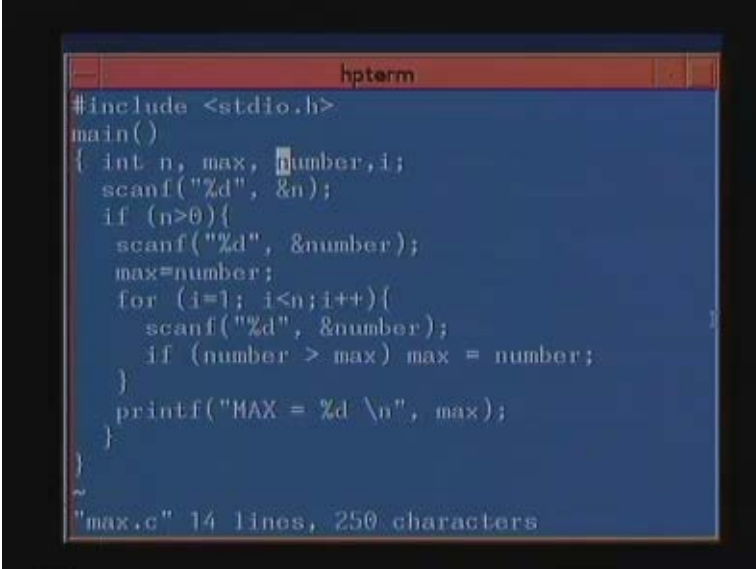
(Refer Slide Time 01:41 min)



So the example that we will take here is to find the maximum of n numbers. Now I had outlined the scheme for finding out the maximum of n numbers also we first read in the number of numbers that you want and then you read the first number, assign it to max and then in a loop you read each number and update MAX if the new number is greater than the current maximum. And that is what we translate exactly into a C program in the next example. So let's go and see the example.

As previously we have got include stdio.h, the main program and this is what the main program ends. We have declared a variable n which will read in the number of numbers, a variable MAX which will store the current maximum and number which is of variable which will, each and every number will be read in to this variable, i will be the loop control statement.

(Refer Slide Time 02:39 min)



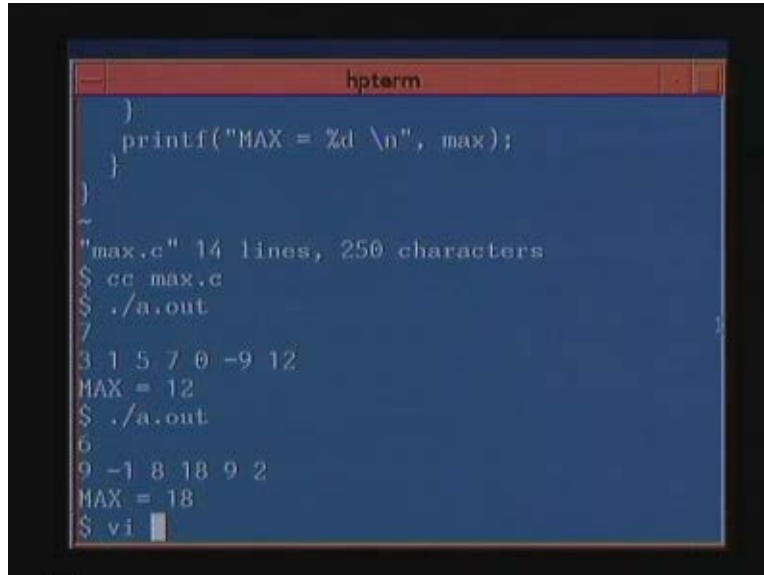
```
hpterm
#include <stdio.h>
main()
{
    int n, max, number, i;
    scanf("%d", &n);
    if (n>0){
        scanf("%d", &number);
        max=number;
        for (i=1; i<n;i++){
            scanf("%d", &number);
            if (number > max) max = number;
        }
        printf("MAX = %d \n", max);
    }
}
~
"max.c" 14 lines, 250 characters
```

So the first statement will be scanf and n. So we will give the total number of numbers which are to be read in. If n is greater than 0, if n is not greater than 0 then obviously we don't have anything to do. So this statement is just there then we read in the first number here and we assign max equal to number in the next line and then here comes the for statement. The syntax is for and there are three parts. This is the first part separated by a semicolon, this is a second part separated by a semicolon and this is the third part. The first part is called initialization, here I have initialized i equal to 1. Here I have done i less than n, this is the conditional. As long as this condition holds the loop will continue. When this condition is not satisfied, it will go out of the loop and i plus plus it just does i equal to i plus 1. There are other issues related to this i plus plus which we shall come to later on but for the time being, it just does i equal to i plus 1. You could have written i equal to i plus 1 here also, I have just written i plus plus to introduce this construct.

And this for loop since there are more than one statement, you have to define it in a block. So the block starts here and ends here. This is the block which has got two statements inside, one is to read in the new number and the other is the if statement, if number is greater than MAX then MAX is assigned number. So this was the program, the algorithm which we discussed in the previous day to find out the maximum of n numbers. And this is the conversion to the program and this is where the for loop ends at this block and then you print MAX equal to percentage d.

Now we are supposed to read in total of n numbers. Now let us check whether we are reading in n numbers. We are reading in one number here and this loop starts with i equal to 1 and continues as long as i is less than n. So it will read n minus 1 numbers in this loop. It will be n minus 1 numbers in this loop. So you can read exactly total of n numbers. So, any question about this program? If there are no questions then let us see how to run it. Let's run it and see. So the first input that you have to give is the number of numbers, I did not write the printf statement. Normally good programming means that you should write the printf statements but since I wanted the program, the whole program to come on one screen I remove this. So let us read in 7 numbers 3 1 5 7 0 -9 and say 12, the maximum is 12.

(Refer Slide Time 06:40 min)



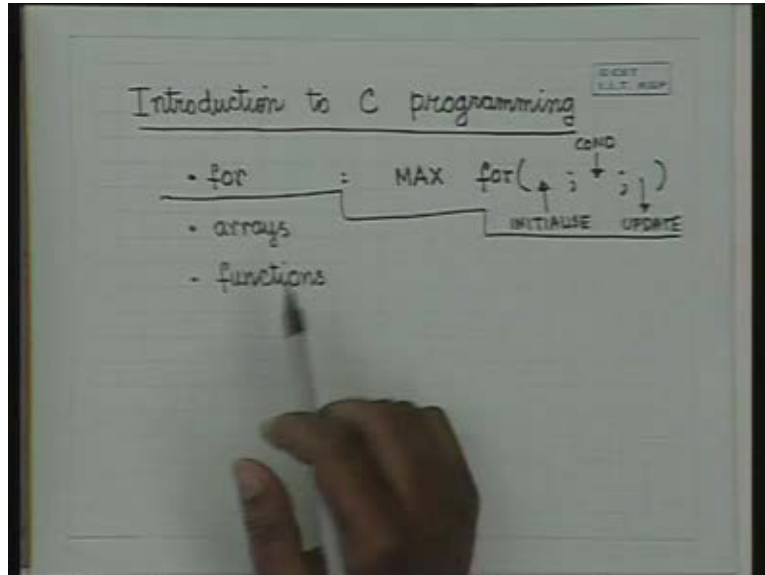
```
hpterm
)
printf("MAX = %d \n", max);
}
)
~
"max.c" 14 lines, 250 characters
$ cc max.c
$ ./a.out
7
3 1 5 7 0 -9 12
MAX = 12
$ ./a.out
6
9 -1 8 18 9 2
MAX = 18
$ vi
```

Let us run it again, say 6 numbers, maximum is 18. So this is how we compute the maximum and this is how we use the for loop. So let's go back to the for loop, so this is the for loop. this is the for loop and we could have even written **from i equal to 2 till i is less than sorry** from i equal to 2 to i is less than or equal to n, till then we will read in n minus 1 numbers inside this loop. So you must write the loop in such a way that it takes care of this, the condition that is taken care of. You can write more than one condition here. you could have written i equal to 1 if you have 2 or 3 variables to initialize, you could have written i equal to 1 and you could have written comma j equal to 0 but the whole initialization must end with semicolon and all the statements inside the initialization will go with commas.

Similarly here you can give multiple conditions but they must be put in one statement like AND OR NOT. And here again you can give more than one statements which will do the loop update. In fact here you could have even read in a number, you could have read in functions also in the initializations like scanf etc but I don't want to go in to complex issues here. So even if you do this i equal to 2, i less than equal to n we still work correctly. So the MAX is 2, so if we just give in one more number, it does not matter or it could have been better to see the other way round. It will just read in the necessary numbers, the rest is just not right. So this is the program to find out the maximum of n numbers.

Here we have used, we have seen how to use the for loop and we have seen that the for loop consists of three parts separated by semicolons, this is the initialization, this is the condition. If this condition is true only then it will go into the loop and this is the loop update that is what are the actions that will take place at the end of the loop. And we have to explicitly update it unlike FORTRAN where you can write do i equal to 1, 10, 2 things like that, automatic update of the variables doesn't take place, you have to explicitly update the variables in a for statement. There are other looping constructs like do and while and we will not explicitly go through them and we will visit them as and when we come through examples.

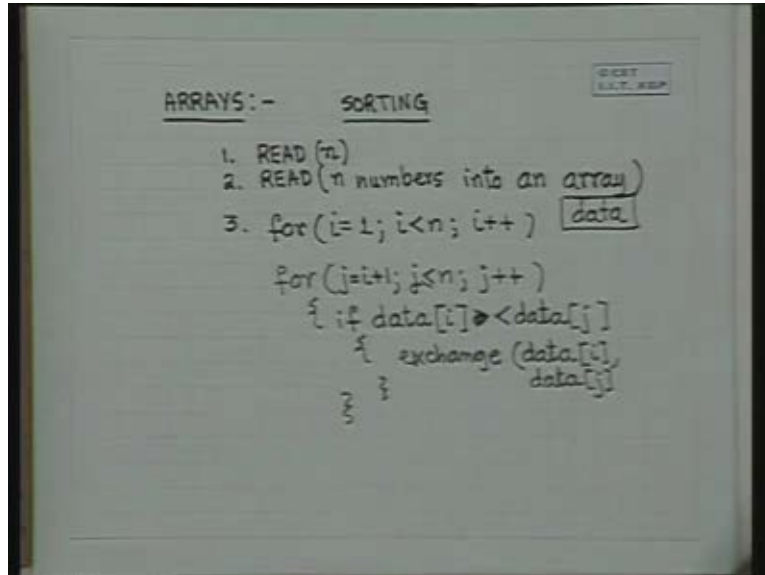
(Refer Slide Time 10:13 min)



So after having done the loop construct, we will see how to use arrays. Arrays are vital for any programming because whenever you have to read in a block of data and you have to use it, you must require arrays. And one of the classical examples of using arrays is sorting. So, we will take up an example of sorting, a simple exchange sort and see how we will use arrays. All of us must have written this program before and the basic algorithm which has been used is READ n then READ n numbers into an array. And then we use two loops to do the sorting and the whole idea behind the sorting program is in each loop, you are finding out the maximum element and storing it in the top position. So we use two loops and in the inner loop, we do a comparison and an exchange. This is the idea of sorting, for example we will do for i equal to 1 till i is less than n i plus plus, when i is equal to 1 we will find out the maximum and store the maximum in the array location a1. So let the array we call it data.

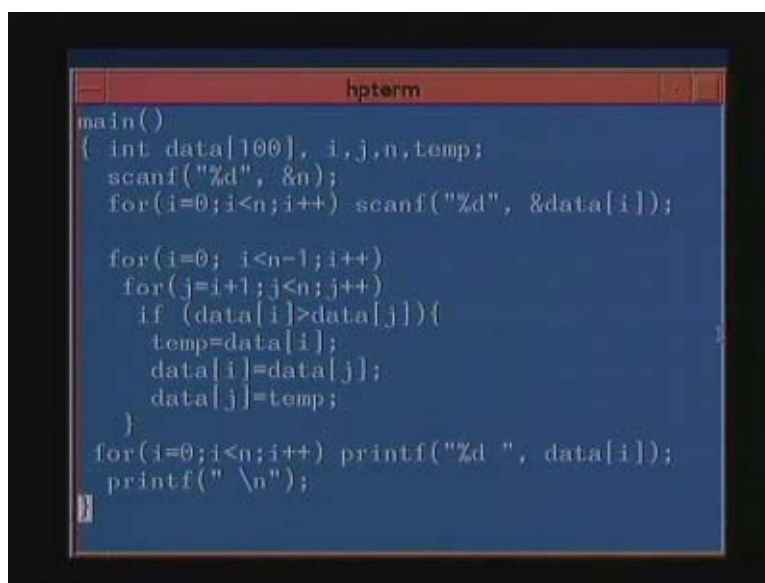
Let the array we call data then when i is equal to 1, we will store the maximum in data one, when i is equal to 2 we will store the second maximum in data 2 that is the maximum from in the rest of the array. So we will use a loop for j is equal to i plus 1 then j is say less than or equal to n and increment it and we will compare **if data i is greater than sorry** if data i is less than data j then we will exchange data i and data j. This is one of the traditional sorting routines. There are other sorting routines like double sort etc. They are similar but slightly different in way of doing it.

(Refer Slide Time 13:55 min)



So this is one sorting routine and we shall see how to write out this sorting routine. The main intention in this program is not how sorting is done because we shall see how sorting is done in a much more better way in subsequent classes but the intention here is to just translate a non-program which contains an array declaration and some for loops into C language syntax. So let us see how the program looks like in our C language. So we include the `stdio.h`, we have just one main program which goes like this from here to here.

(Refer Slide Time 14:19 min)



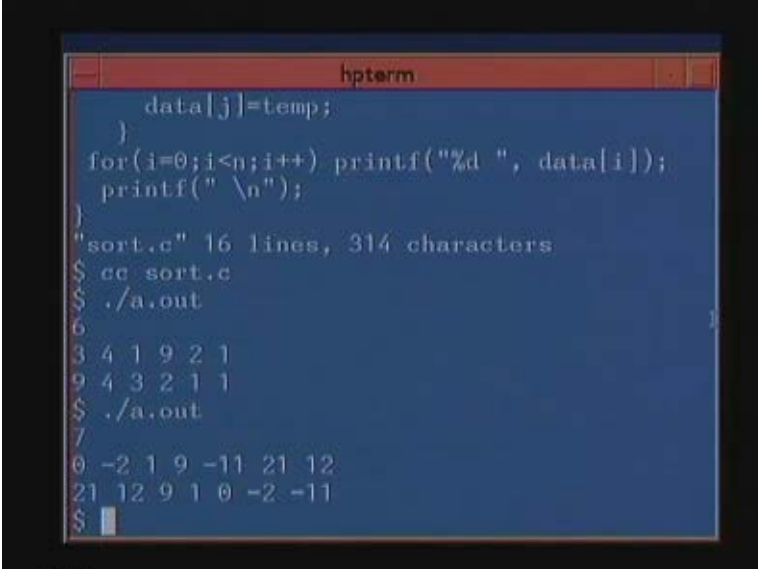
Now this is where we have declared the data array, array data. so `int data 100` this means the data is an array of size 100 and it is an array of integers, the type is integer, `i j n` and `temp` these are

the variables which are declared. The first thing I do is read in the number of numbers and then in a loop, I read in all the elements of the array. An interestingly I have read in from 0 to n minus 1, though we saw in just the minute ago that we were working from 1 to n, in C language when we declare an array it starts from 0. So if we declare an array data 100, its starts from data 0 to data 99, not from data 1 to data 100. So these **zero location** 100 locations start from 0 to 99 rather than 1 to 100.

So you can utilize the zeroth location so that is what I did here, for i equal to 0 i less than n i plus plus and since this is a single statement, I did not require to put it in braces because this statement is the only statement in the loop, so I did not require to put the brackets but if you put the brackets also its okay. We will put two brackets here and the closing bracket after this. So scanf I read in integers and AND data i, I pass the address of the location data i. And then here is the for loop, we have previously discussed i equal to 1 to i less than n, since we are going from 0 means i equal to 0 to i less than n minus 1 i plus plus, j equal to i plus 1 to j less than n j plus plus. If data i is greater than data j then we exchange it. So this one actually does finding the minimum, if you want to find the maximum we will have to change this to less than.

So data i is less than data j, we exchange and we use these three statements to exchange. We use the temporary variable, assign it to data i, data i is assign to data j and data j is assigned to 10. and this is the loop not that for this for statement, we did not require opening braces and closing braces because this for statement has only **one for**, one statement for it and that is this for statement. So we did not require an opening bracket here and a closing bracket here after this because this for statement has only one statement which is another for statement and this for statement has an if statement, has a statement which is only one statement which is an if statement and this statement has got a set of blocks. So it is just sufficient to put brackets here, though if you put brackets here as well as here, you would have been safe but that is not necessary. And finally we printed out the sorted array i equal to 0, i less than n, i plus plus printf and then at the end of after printing all of them, I printed a new line and this terminates the C program. So, any questions on the program? So then let's run. The first we have to give is the number of numbers says 6 and then all of them.

(Refer Slide Time 19:17 min)




```
data[j]=temp;
}
for(i=0;i<n;i++) printf("%d ", data[i]);
printf(" \n");
}
"sort.c" 16 lines, 314 characters
$ cc sort.c
$ ./a.out
6
3 4 1 9 2 1
9 4 3 2 1 1
$ ./a.out
7
0 -2 1 9 -11 21 12
21 12 9 1 0 -2 -11
$
```

So it prints the sorted array 9 4 3 2 1 1, 7 numbers. So this is how it gets printed. So, this is the program for sorting and this if I want to print all the numbers one after another then I would have to give a backslash n out here. If I give a backslash n out here then the numbers would come in one all different lines. For example if I give a backslash n out here, so I have given a backslash n here. So when I print the numbers, they are all supposed to come in different lines. So this is all you can use the controls to separate out. We shall comeback to this program and study later to understand two things. We will come to a variant problem and we will see whether how we can solve that problem better. And we will see whether sorting is to be done this way at all or not, whether there is a better way to do sorting. And we shall go back to our old steps of initial problem definition, refinement and generation of final solution to see whether we can get a better program for sorting.

And why, this is not the best program for sorting. So we shall come back to them later on because it is our intention here to just go through arrays and functions. The next example that we will take is that of a multidimensional array. We shall see how to declare the 2 D array and how to do matrix multiplication. So this is an example of matrix multiplication where we will do multiplication of two arrays of integers.



(Refer Slide Time 21:34 min)

A screenshot of a terminal window titled 'hpterm' with a blue background. The code is written in white text. It includes a header file, declares three 10x10 integer arrays (a, b, c), and two sets of nested loops for reading data and performing matrix multiplication. The status bar at the bottom indicates the file is 'matmul.c' with 31 lines and 558 characters.

```
hpterm
#include <stdio.h>
main()
{ int a[10][10], b[10][10], c[10][10];
  int i,j,k,m,n,o;

  /* Reading the Data */
  scanf("%d%d%d", &m, &n, &o);
  for(i=1;i<=m;i++)
  for(j=1;j<=n;j++)
    scanf("%d", &a[i][j]);

  for(i=1;i<=n;i++)
  for(j=1;j<=o;j++)
    scanf("%d", &b[i][j]);

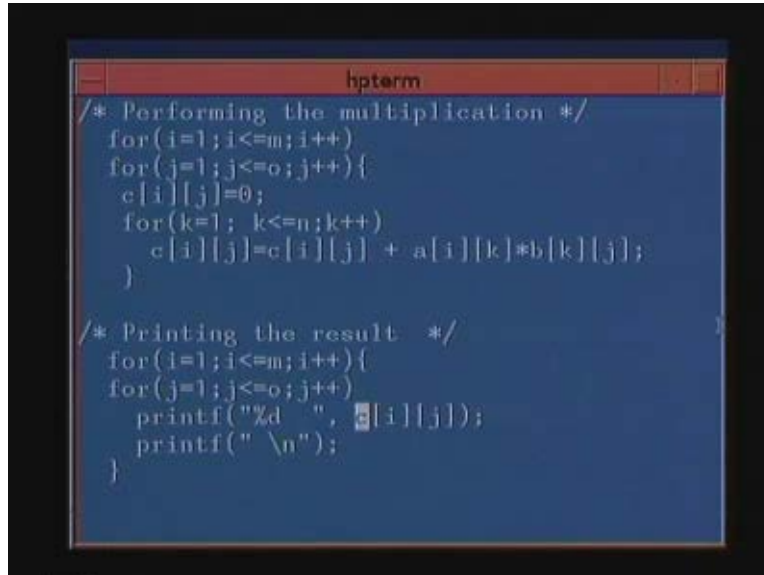
"matmul.c" 31 lines, 558 characters
```

Now here 10 by 10 array of integers is declared like this, a [10] [10]. What is the advantage of doing it this way? I will come and explain a bit later, b [10] [10] c [10] [10]. We shall go deeper into why such things are there to appreciate, why C language has changed the constructs instead of writing a 10, 10 where have they done it this way. What does it mean, what does all signify, we shall come back when we got used to more of C programming. So we have declared some integers, we read in the data so we first read in the three dimensions because we have to read in two arrays. One is an m by n array and another is an n by o array. And finally if a is an m cross n array and b is an n cross o array then c will be an m cross o array. So we will first read in the dimension m n and o and after that we read in the data. Here we have read in the array a in a double loop, i equal to 1, i less than equal to m but it would have been appropriate to actually do it from 0 to m minus 1 because somebody gives you 10 by 10 then you would land up in problem because these arrays are always going to be declared from 0 to 9.

Anyway we will make sure that we don't give the data that way. So i equal to 1, i less than equal to m, i plus plus, j equal to 1 so we are reading it row wise and again here AND a [i] [j] and we give it this way a [i] [j], not a i, j you give it this way. Again we read in the array b and exactly the similar way except that the constructs are n and o because we are reading in by n by o array. And this is where we perform the matrix multiplication; I don't want to go into details because all of us possibly understand how to do matrix multiplication. There is an auto loop i equal to 1 to m, since m by n and n by o, so we will get an m by o array. So c array, the loop in this here will be from i will be m and j will be o and the first thing we do is initialize c [i] [j] to 0. And for k equal to 1 to n we do c [i] [j] is equal to c [i] [j] plus a [i] [k] into b [k] [j]. This is a standard matrix multiplication. This is just an example to use two dimensional arrays in C language.



(Refer Slide Time 25:04 min)

A screenshot of a terminal window titled 'hpterm' with a dark blue background and white text. The code is as follows:

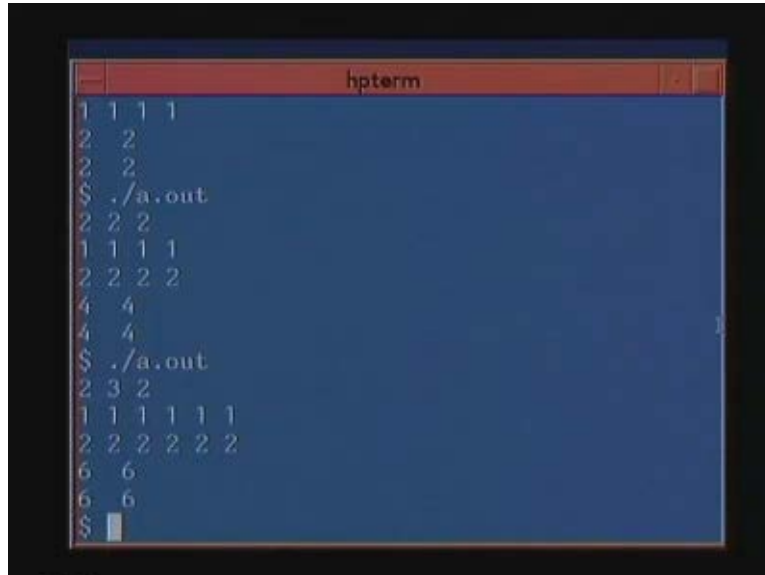
```
/* Performing the multiplication */
for(i=1;i<=m;i++)
for(j=1;j<=o;j++){
c[i][j]=0;
for(k=1;k<=n;k++)
c[i][j]=c[i][j] + a[i][k]*b[k][j];
}

/* Printing the result */
for(i=1;i<=m;i++){
for(j=1;j<=o;j++)
printf("%d ", c[i][j]);
printf("\n");
}
```

And here we print the result. And to print the result we have printed it in row wise format i is equal to m or 1 to m, j is 1 to o and after every row see, this statement means for here, this means from here to here. So this is the block, this block contains one for statement which is this. For i, actually this for statement actually goes like you can write it in separate lines but this is how it is. this is one for statement, for j equal to 1 j less than equal to o j plus plus you print this and at the end of this whole loop, you print a new line.

Even if you write it this way, it does not matter. The brackets absolutely take care of what your intention is. So first we have to give the three, so let's give a simple example for all of them are equal. Then you have to give the first one, so in order that we understand the result properly let's give a simple example. So if we have given an array of two matrices all are one's, 2 by 2 matrices obviously we are supposed to get back **two's**. Let's give another example. Am sure we can make out that the results are correct, the example, the inputs are given in that way.

(Refer Slide Time 26:56 min)

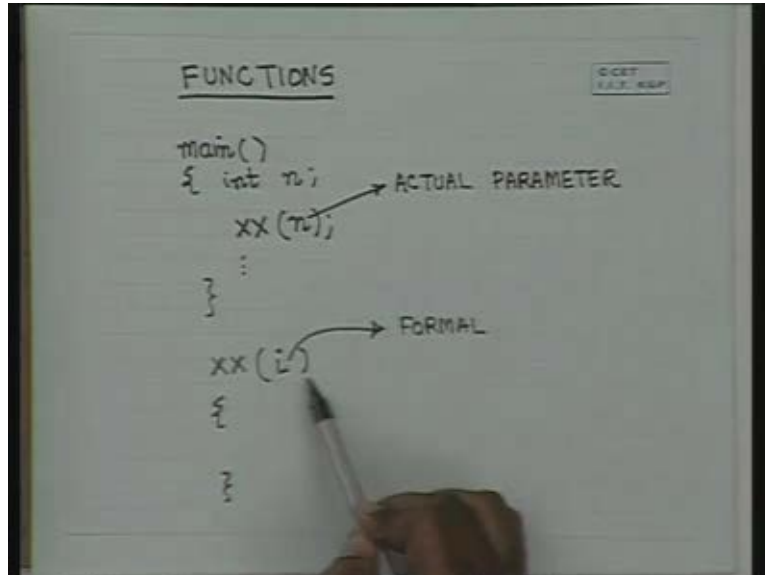


```
hpterm
1 1 1 1
2 2
2 2
$ ./a.out
2 2 2
1 1 1 1
2 2 2 2
4 4
4 4
$ ./a.out
2 3 2
1 1 1 1 1 1
2 2 2 2 2 2
6 6
6 6
$
```

So, now given a 2 by 3 and a 3 by 2 and I am expected to get a 2 by 2 back. So the first one in order that we get the results, understand the results correctly let us give 6 1's, in the next one let us give another 6 2's. So, obviously the program works correctly and the intention was not to understand how matrix multiplication is done but how to declare two dimensional arrays, how to put in for statements and how to put in if then and for, combination of the if then and for in a two dimensional array. So that takes care of two dimensional arrays and once we know what two dimensional arrays are we can easily extend it to 3 4 5 dimensional array in a simple and a similar manner. But we shall come back to these examples to understand why we do it this way and what advantages these lead us to; we shall come back to these examples later on.

The next we come back to a very very crucial concept, functions that is user defined functions. Now in any program we have shown that we know some programming languages like FORTRAN or basic or Pascal and in all of these languages, you can use functions. So you will in a program, see in a C program you will have a main program and maybe you will have an integer n and maybe you will call a function say xx with argument n and then do some other things and terminate. And we will declare a function xx with argument i and then you will do something in this function. It may be a function; some people call it subroutine also because you may not return a value. In C language everything is a function and you can return values through the arguments in a special kind of a way.

(Refer Slide Time 30:08 min)



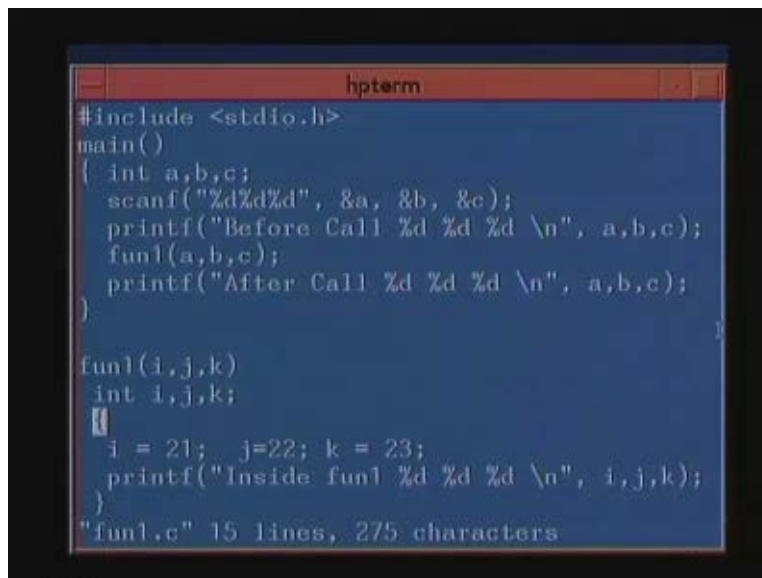
Before we move into C language examples, let us clarify some terms. There are two parameters of a function, one is called the actual parameter and this is called the formal parameter because this `xx` may be called many times with different actual value, actual arguments and this is the formal parameter, wherever the function itself is declared. The type of the two must match; if the types do not match there will be problems. Obviously you can have more than one parameters and here also you must have the same number of parameters, you cannot have less or more. And their type should match in a one to one fashion, this is well known. So now let us see through an example, what happens in a C program function. So let's take a very very simple example.

(Refer Slide Time 30:38 min)



The first example will help us to understand the syntax of calling a C function as well as the way a function behaves. So here I have got, now I have got a main program which ends here and I have got a function called fun1 which ends here. And now what do I do in the main program? I do `int a b c` then I read in three values `scanf` and `a b` and `c` and in these three values before the call I print what the values are, I call the function like this `fun1 (a, b, c)`. I can call it, I can do an assignment also `x equal to fun1` but we will see that later on. And then after the call, I will again print the value of the variables. And inside the function corresponding to `a` I have `i`, to `b` I have `j` and to `c` I have `k` and before the brackets open for the function, I have to declare the type of the argument.

(Refer Slide Time 31:45 min)



```
hpterm
#include <stdio.h>
main()
{ int a,b,c;
  scanf("%d%d%d", &a, &b, &c);
  printf("Before Call %d %d %d \n", a,b,c);
  fun1(a,b,c);
  printf("After Call %d %d %d \n", a,b,c);
}

fun1(i,j,k)
int i,j,k;
{
  i = 21; j=22; k = 23;
  printf("Inside fun1 %d %d %d \n", i,j,k);
}
"fun1.c" 15 lines, 275 characters
```

I can declare it here also inside the declaration, I can also declare it below. So I will use the approach of declaring it below, we will see examples where we declare it. So I declare them to be integers and inside the function I assign `i` equal to 21, `j` equal to 22 and `k` equal to 23. I need not put them in separate lines, just semicolon takes care so. So that's fine and I put in the print statement that inside `fun1` `i j` and `k` are these, alright. So this is the very simple program to understand what this syntax of a function definition is. The syntax is the first the function name the argument then the type and then the body and the body again comes in brackets like this, just like a main program. And here I have just read in 3 values and before the function is called, I printed it. I called the function, inside the function I printed it, after assigning it this is what I have done.

If such an equivalent program was written in FORTRAN then whatever I would have read would have been printed here, inside the function they would have been converted to 21 22 23. And after the function call they would have been converted to 21 22 23 but that's not what will happen in C. So I read in three values 2 3 5. Now what do I get? I get before the call it is 2 3 5 that is what I have read in, inside the function obviously after assigning it to 21 22 23 and after the call it is 2 3 5. So I did not get a change when I returned back. That is before the call

whatever I read in I got, in the function these were reassigned and after the function they were returned back.

So, why I did not get a change is the first question and if I do not get a change then what is the use of calling such a function or how would I get a change if I want such a change are the question that we will have to answer. Because if a function does just some, it requires some values and I may want some value to be updated. How would I do it? So to answer this question first we will have to see how function parameters are passed in C, what are the ways in which function parameters can be passed and how do I get back values in C, if I want to pass the function parameters. That will also give us some hint about why scanf uses this AND.

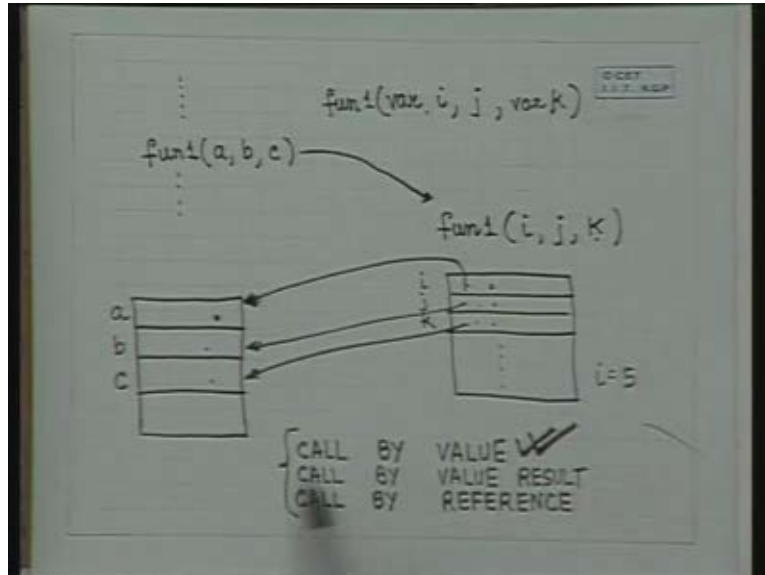
So let's go back to our workbook and see what we can do. So we have got the actual parameter here and the formal parameter here. Now in any programming language, we shall see that when there is a function call say fun1 a b c, there is a call and we have got the actual formal parameters as I mentioned before i j and k. Now these a b and c are occupying some locations in the memory, the schematically let us say a b and c. Now when this function is called, a new data segment for this function is set up, the variables, the arguments as well as the local variables of this function are allocated certain amount of memory of which other than the local variables i j and k are also allocated some memory locations.

In C language this is what exactly happens. When a function is called, this memory is allocated. The value of a is copied into the value of i, the value of b is copied into the value of j, the value of c is copied into the value of k and then this starts executing. After the execution is over, this memory is de-allocated and the values are not copied back. This is how C programming works, this is called call by value. This is called parameter passing using call by value, C uses call by value. So they are copied into the local variables and the function is executed and after the function is over, these are de-allocated. We shall see how this allocation and de-allocation and memory access is done little later on in more involved examples. In other programming languages there are slight variations, there is a variation called call by value result.

What this does is when the function is called, it copies the parameters, arguments execute these function and after that execution before de-allocation it copies it back, copies the new value's back. So that is called call by value result. In the other one which is used in FORTRAN is called call by reference. What happens here is instead of these values, these locations are not here or these are not actual locations but these contain addresses to this location. And whatever you do here, whenever you update i here this address is taken and this value is modified. The modification is done on the content of this address here, alright.

So if you write i equal to 5 then here there is a change. It uses this location to keep the address of a, it gives us this location to keep the address of b and gives us this location to keep the address of C and it uses these address to modify the contents. So there is star of i operation whenever an i operation is going. So if you want to change something in i in the function here, it uses the content of this, it modifies the contents of the address which is something called indirect addressing but this is what is called call by reference. There are other ways of parameter passing like call by name and others but we will now concentrate on call by value, call by value result and call by reference and C language uses call by value.

(Refer Slide Time 42:11 min)



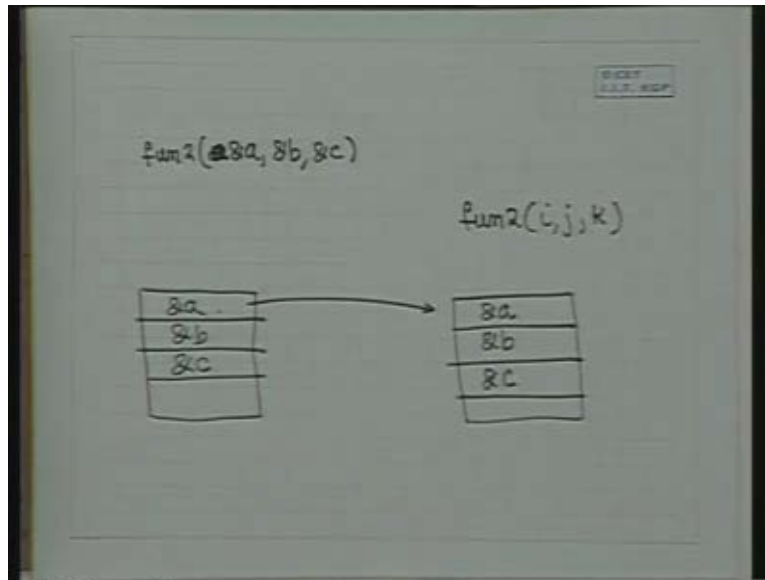
So, C language uses call by value and not call by value result or call by reference. Anyone who knows Pascal will know that in Pascal, you can define this parameters in another way. If you write `fun1 var i` then this means that this will be used as call by reference. If you write simple `j` it means call by value and again if you write say `var k` then it will mean call by reference. So in Pascal if you have written `var i j` and `k` and done this then this would not have changed but this and this would have changed, whereas in C everything is uniform call by value, so we will have to see next how we make it call by reference. To make it call by reference, what we do is we pass the address location and modify the contents, alright.

So let's see in this example how we will convert it to call by value. So here what we did was we read in `a b` and `c`, this is the value of `a b` and `c` and we passed in the two `fun 2`, we have used another function called `fun2`, we have passed the address `a AND a`, `AND b` and `AND c` and printed the values after the variables. And in `fun2` we declare `i j` and `k` but we now have to declare these as the address locations rather than variables. So this is how we declare it `int star i`, `star j`, `star k`. To understand what this means, a very simple logic has to be applied. What does `star i` mean, **i meant** `star i` means the content of `i`. So it says that the content of `i` is an integer. This means the content of `j` is an integer and this means that the content of `k` is an integer. Understanding it this way will take us one step ahead in understanding all declarations in C where lot of infringing issues for declaration in C. We shall see them slowly but here what we have done is we have done this and we have done `star i equal to 21`, `star j equal to 22`, `star k equal to 23`. We have printed the values of `star i`, `star j` and `star k` and comma.

Now let us see what should happen. Let's go back and just work it out first and then we will see. So we had a call now `fun2 a` where we passed `AND a AND b AND c` and here we had `fun2 i j k`. So the three locations which will be copied here into `i`, here `I` will have the address of `a` in `i`, here all have address of `b` and here `I` have the address of `c`. So these address of `a` which is here will be copied in here, address of `b` will be copied in here and address of `c` will be copied in here. These are the parameters. What is the value of `AND a`? `AND a` means the address of `b`.

So once this is copied in here, it does not matter whether it is copied back or not. What I am going to do is I am going to modify the contents inside.

(Refer Slide Time 45:28 min)



So those contents are going to change anyway and I am going to get it back reflected. So that's exactly, so we will just quickly see the program. So what I have done is I have passed the addresses `a b c`, change the contents and so now if we run it, so now before the call it is `1 2 3` inside the function and now I have though its call by value, I have made sure that after the call I get back the correct values. And now you will appreciate why in `scanf` we have to pass the addresses because its call by value, if you are passed `a b` and `c` whatever you read inside wouldn't have been reflected back when you returned.

So that is why you passed the addresses and the `scanf` function actually modifies the contents. So whenever you want to get values back and passed back, we will have to use the contents and the address locations in C. This is a typical C style of programming, it's unlike in other languages like Pascal where it is like it different in FORTRAN its pure call by reference, in C its call by value and if you want reference calling we have to do this way. So we will stop here today and we will see more details of how to do function parameter passing and other things in the next class. Thank you.