Programming & Data Structures

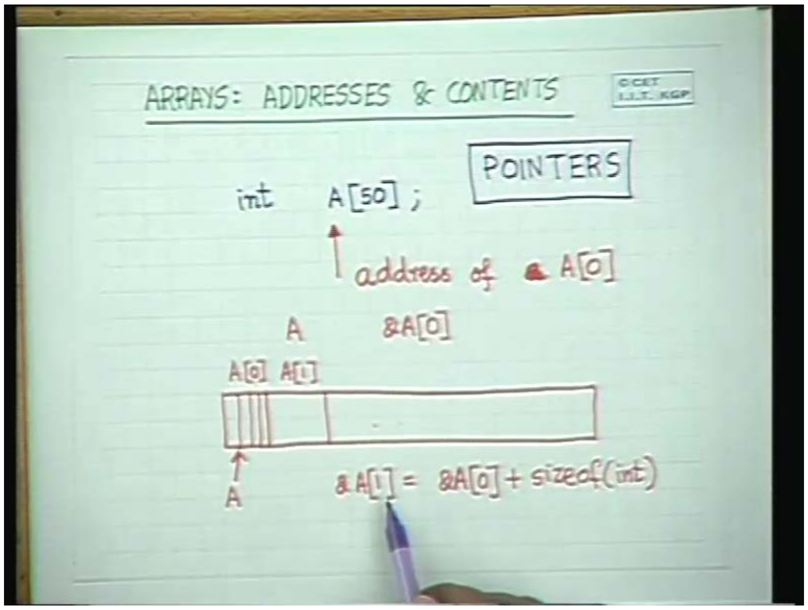
Dr. P.P. Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture 13

Arrays: Addresses and Contents

In this lecture, we will study how arrays are stored and how arrays are accessed. We have already used arrays, character strings and all that but there are certain other details about arrays which we need to understand. And we will understand them through several programs and using these programs, we will be able to see what they mean. Till now we already know that you can declare an array, an integer array say A like this, a one dimensional array A can be declared like this.

(Refer Slide Time: 05:15)

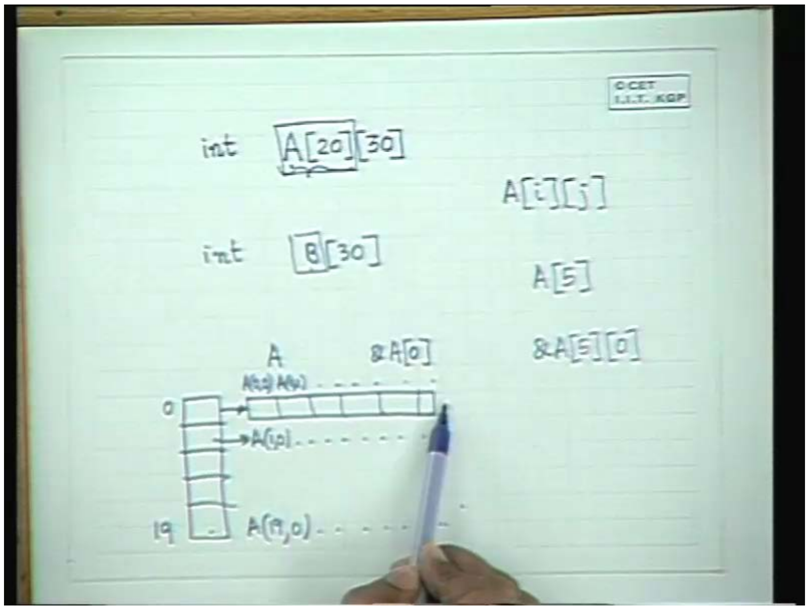


A is the array name and 50 elements are there of type, each of type one integer. So, 50 elements each of type one integer is stored in this array A. Now let us try and understand the meaning of certain things. When you write int A [50], it means that A [i] the content that is A[i] is of type integer, the content of A[i] is an integer. Please understand this very carefully. The content of A[i], the content of A is 0 to A 49 are all integers.

A, this A is the array name and can be used in various capacities. This is the address of… So A, the value of A is an address and it is equivalent to this address, this is the first point. The second point is that the array is stored in contiguous locations in the memory, if this is A [0] which is also the value of the address A, that is you can access this A as an address.

Then the next will be A[1] but they are not one after another because if this requires 4 bytes of memory then this is after 4 places this will start in the memory. So A[1], address of A[1] will be address of A[0] plus the size of an integer. There is a size of function which you can have a look at and find out sizes of various pre declared types. So this is plus 1 and so on. So, one dimensional array is stored in contiguous locations like this. In C these addresses have values and these addresses are also known by a very popular term called pointers. They are pointers to locations; addresses of locations are also called pointers to locations. So A is a pointer to the start of the array and A[1] is the pointer to the location A[1] that is how we refer to it in terms of C language.

(Refer Slide Time: 09:10)



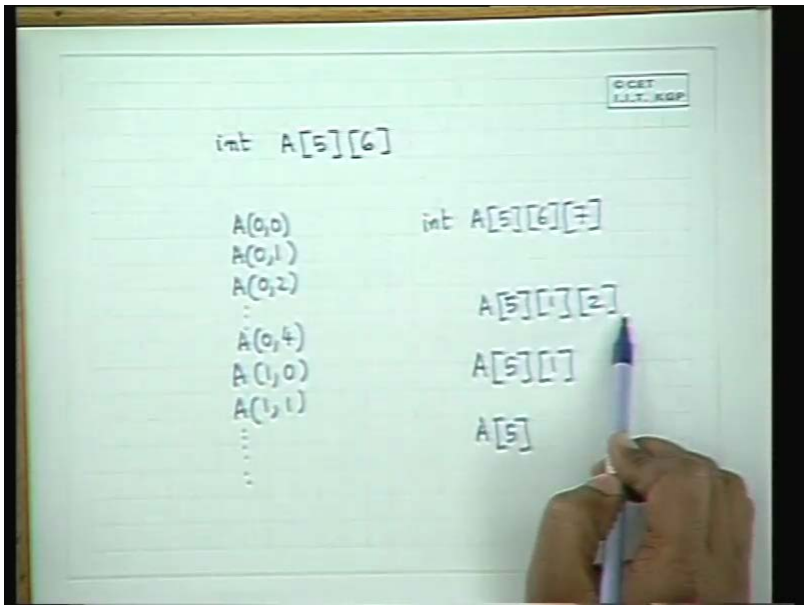
Second point let us come to a two dimensional array. A two dimensional array say written like this, it means that the content of any A [i], i between 0 to 19 the content of any A [i][j], i between 0 and 19 and j between 0 and 21 is an integer, alright. Now see this int B[30], B there is a pattern in this whole definition. B is the address of the start of this array.

Similarly A is the address of the start of the two dimensional array. So A is the same as the AND A[0] and A[5] suppose I say A[5], what do I mean by A[5]? A[5] means this is this part of the array, A[5]. For example this part B is the starting location of the array B which is the address of B[0]. A[5] will be the starting location will be equivalent to AND A[5] [0], it will be the starting location of the array starting with A, so conceptually 0 to 19 and then you can address it like this. This is one way of looking at it, A(0,0), A(0,1) so on, A(1,0) and so on, A(19,0) and so on. This is one point that is A[5] is the address. Since the content is this, part of it will give you the address.

This will give you a two dimension A[i] will give you a one dimensional array but each array, each element of that one dimensional array is an array is another one dimensional array that is the idea. Each element of this one dimensional array is a one dimensional array. A is an array and each element of that array A is an address which considers such that each element of this block is a two dimensional. A is a pointer to a two dimensional array, this is a pointer to a one dimensional array and this gives you the final location, alright.

Say this is so, you can conceptually have a look at it this in this way. These are the addresses of one dimensional array. The second point is how they are stored in the memory? suppose you declare int a five six then they are stored in the memory this way, the contiguous storage in the memory is A(0,0) followed by A(0,1) A(0,2) A(0,4) then A(1,0).

(Refer Slide Time: 11:30)



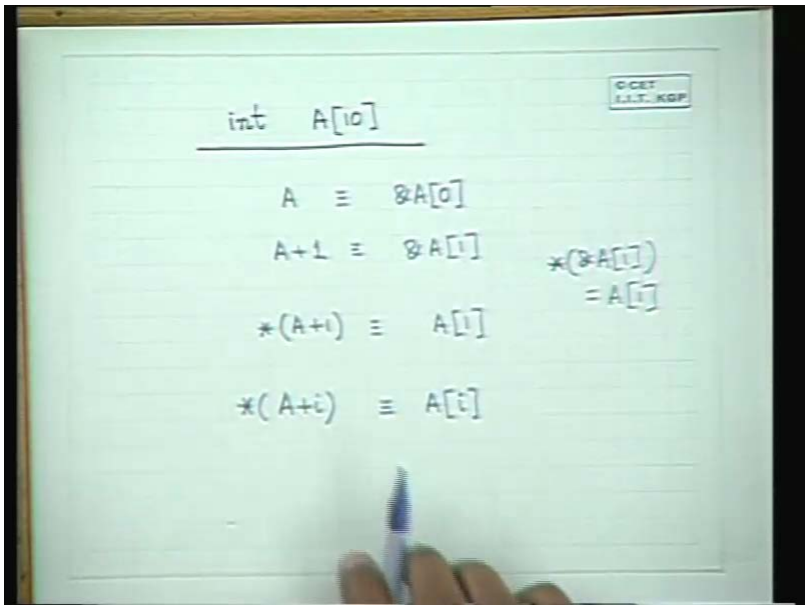
©CET I.I.T. KGP

```
int A[5][6]

A(0,0)            int A[5][6][7]
A(0,1)
A(0,2)
  :                 A[5][1][2]
A(0,4)
A(1,0)              A[5][1]
A(1,1)
  :                 A[5]
  :
```

This is how they are stored in contiguous locations of the memory. We will run programs to just check out, you know you can just print out addresses and you can just check what the exact values are. So it will not be difficult to see what they mean. So this is the sequence of contiguous locations and if you declare, then this is a three dimension array and if you write down A[5] [1] [2] then it will give you that value of the location 5 1 2. If you write A[5] [1], it will give you the address of the array of two elements which starts at 5 1. So you can look at it, you can look at it as a multi-dimensional structure. And if you write down A[5], it will give you the address of the 1 by 2 array from the 5th place. So you look it as a cube and if you look at it as a cube then you will see then inside this cube you have got sub cubes.

This three dimensional array is of 5 elements, each element is a two dimensional array like this. And there are 5, suppose this was say 2, there are 10 such blocks of which each of them 0 0 0 1 up to 5 2 each of them is a one dimensional array. And there are 20 such blocks where each of them is an integer. That is the way you have to look at the array structure. This will give you several ideas about how to declare arrays, how to call arrays because they can be accessed in various ways.

(Refer Slide Time: 13:35)



©CET I.I.T. KGP

int A[10]

$A \equiv \&A[0]$

$A+1 \equiv \&A[1]$

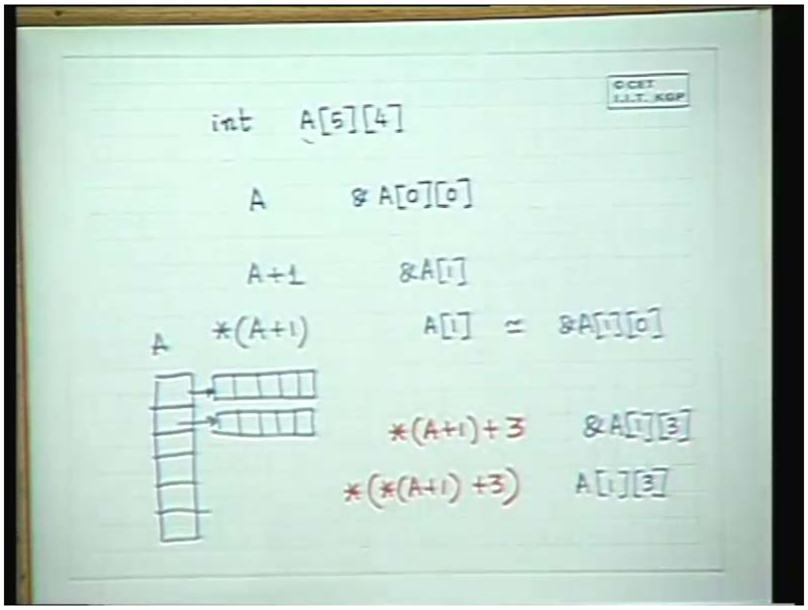
$*(\&A[1]) = A[1]$

$*(A+1) \equiv A[1]$

$*(A+i) \equiv A[i]$

Secondly there is a content and the address location int A[10]. Suppose I declare this, then A is the same as AND A[0], A plus 1 is the same as AND A[1]. Content of A plus 1 is the same as A[1], the content of the location AND A[1], star of AND A[1] is A[1]. So star of A plus 1 is A[1], content of A plus 1 is A[1]. So this way A plus i content is the same as the value of A[i]. So you can use this also to access an array. Coming to two dimension arrays, A is the same as AND A, A plus 1. What is A, what is A plus 1 that is the first question. What is A plus 1, what is the content of A plus 1?

(Refer Slide Time: 16:53)

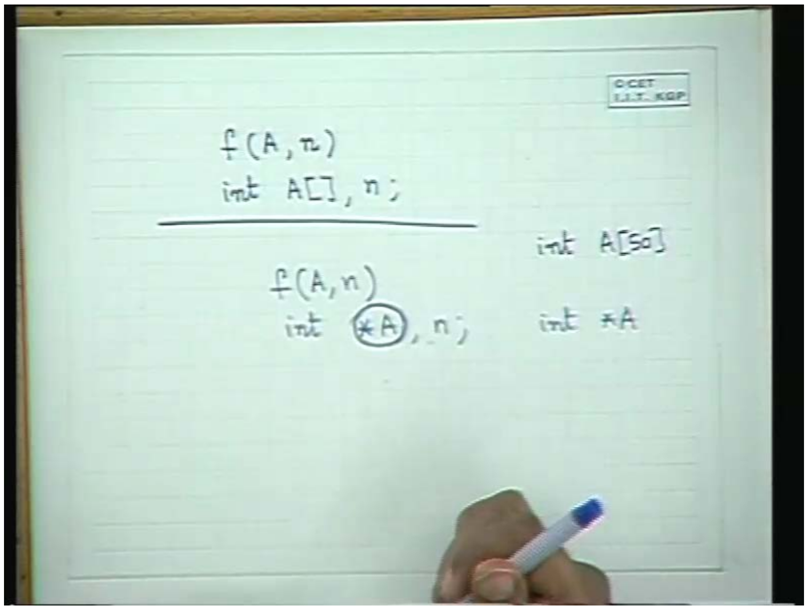


The content of a plus one is A[1], this is the same. If it was a one dimensional array a plus 1 would have been AND A[1], here also it is AND A[1]. This is very streamlined, A plus 1 is the same as AND A[1], content of A plus 1 is a one but since it is a two dimensional array, it is equivalent to right… This is the starting location, A[1] is the starting location of a array of 4 elements. This one look at it conceptually like this.

This is the array A, A[0] A[1] A[2] A[3] A[4] and then each of them bifurcates into the two dimensional array. So this is A, this is A(1,1) and if you have a three dimensional array then each of them will again go out and make arrays. And if you do not understand this structure carefully, you will not appreciate the address and the content configurations in C. So this structure must be understood very clearly and carefully. So what is star (A plus 1) plus 3 star, it is the content address of the location A [1] [3] and star of that will give you the value of the, will give you the actual location the value because now this ends up in a slot which is an integer. That is how it has been defined. This is the address of the two dimension array, this is the address of the one dimension array, each element composing of an array and this is the address were each element consists of a slot. So this way you can do addressing and get the contents of variables.

There are other declarations of addresses or pointers which we shall see later on when we see dynamic allocation except in arrays where you don't have to give the array dimension. For example in a function f where a is an array and n is an integer, we used to declare int a and you don't have to give the dimension because you don't have to allocate elements.

(Refer Slide Time: 19:05)

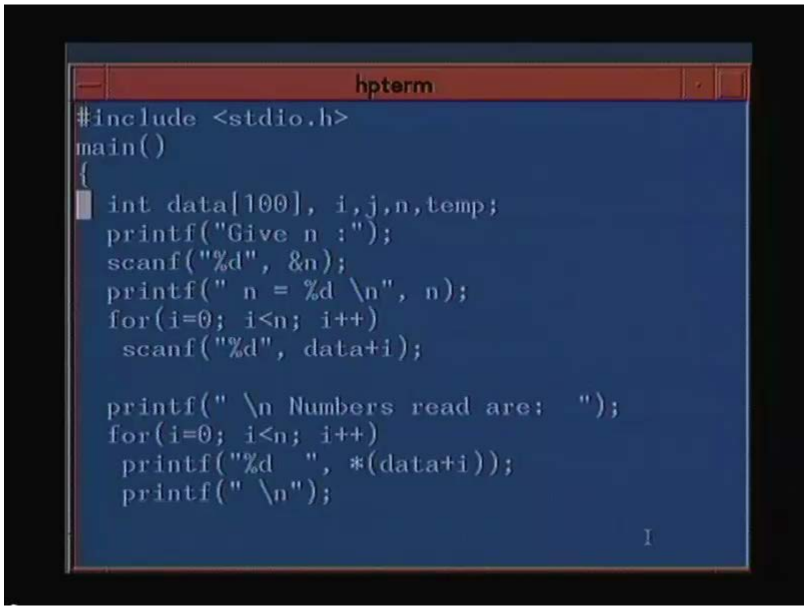

©CET
I.I.T. KGP
f (A, n)
int A[], n;
int A[50]
f(A, n)
int *A, n;
int *A

Here you can also do f (A, n) int star A, n which means that the content of A is an integer and then you can, A plus 1 you will get the address of that location but in such a situation only one pointer element will be allocated as a variable and you can move this pointer along this array because you have just passed the array name but you have not allocated the array values. This has been, this will come as it comes through parameter passing but in the main program if you want to declare an array say int A 50 and instead of that you declare int star A then it will not automatically work because here you have allocated fifty elements, here you have only allocated a pointer to the array.

In order to allocate another fifty elements during execution of the program, you have to do something else, you have to do dynamic allocation which we shall come to later on. For the time being we are not bothered about dynamic allocation, we are just trying to see what arrays and structures mean, sorry arrays and pointers and contents of arrays mean. So we will go through a set of 5 examples which will show us how they can be used in a variety of ways.

This is a bubble sorting or exchange sorting program. So in this exchange sorting program we read in a set of elements, we do sorting by exchange sort and we try to introduce various concepts of addresses.

(Refer Slide Time: 19:40)

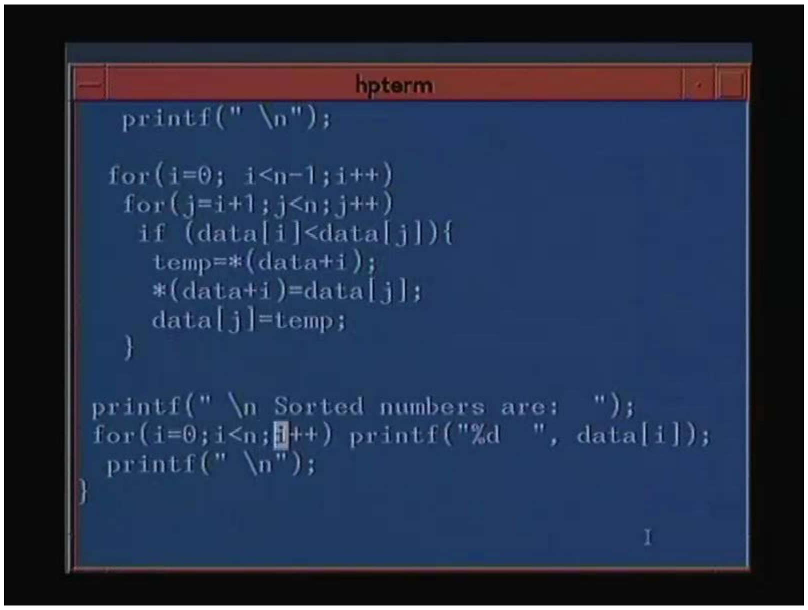


hpterm

```
#include <stdio.h>
main()
{
  int data[100], i,j,n,temp;
  printf("Give n :");
  scanf("%d", &n);
  printf(" n = %d \n", n);
  for(i=0; i<n; i++)
   scanf("%d", data+i);

  printf(" \n Numbers read are:  ");
  for(i=0; i<n; i++)
   printf("%d  ", *(data+i));
   printf(" \n");
```

The first one we declare integer data [100], i, j and temp, read the value of n, print the value of n and then read the data. Now here normally we used to give AND data i but here we can also give data plus i which is the same as AND data i. And instead of printing data i, we can print it by star of data plus i. So you can just read and print, you will get the values. And sorting, this is a simple sorting routine for i equal to 0 till i is less than n minus 1, j equal to i plus 1 j less than n. If data i is less than data j, here again instead of data i you can write star of data plus i.

(Refer Slide Time: 21:35)



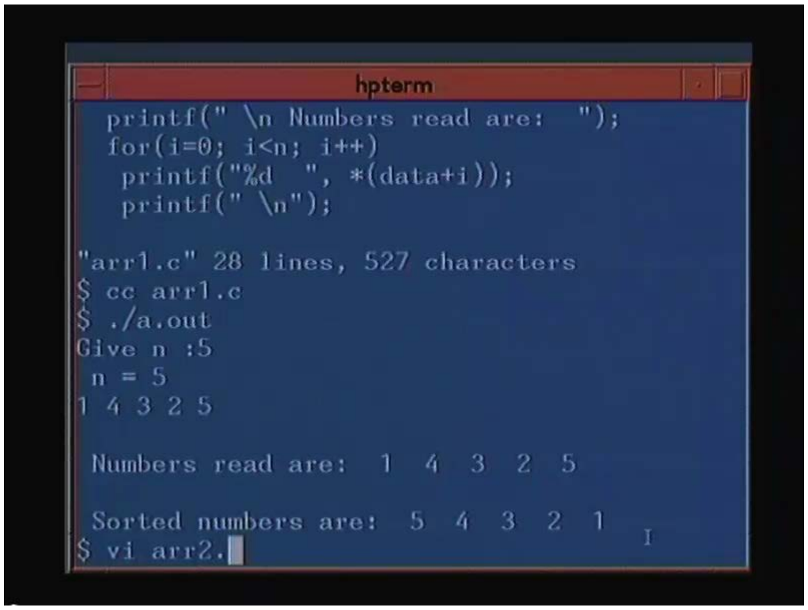
hpterm

```
   printf(" \n");

  for(i=0; i<n-1;i++)
   for(j=i+1;j<n;j++)
    if (data[i]<data[j]){
     temp=*(data+i);
     *(data+i)=data[j];
     data[j]=temp;
   }

 printf(" \n Sorted numbers are:  ");
 for(i=0;i<n;i++) printf("%d  ", data[i]);
  printf(" \n");
}
```

On the left side also you can write star of data plus i but on the left side, you cannot assign data plus i equal to 5 that you cannot do. You cannot change the array address. Suddenly if you want to change the address of the array, the array is allocated by the machine. If you want to change the array address, you cannot do that. Unless you declare a variable of pointer type, you can change that variable. All variables that you declare for example here, the contents of the variables i can change I cannot change the address of the variable because the addresses are allocated by the machine. Therefore here star data plus i equal to data j and data j equal to temp.

And again we can print them here, you print data i you can print star of data plus i. So we can use them in such a variety of ways and it will work quite fine.

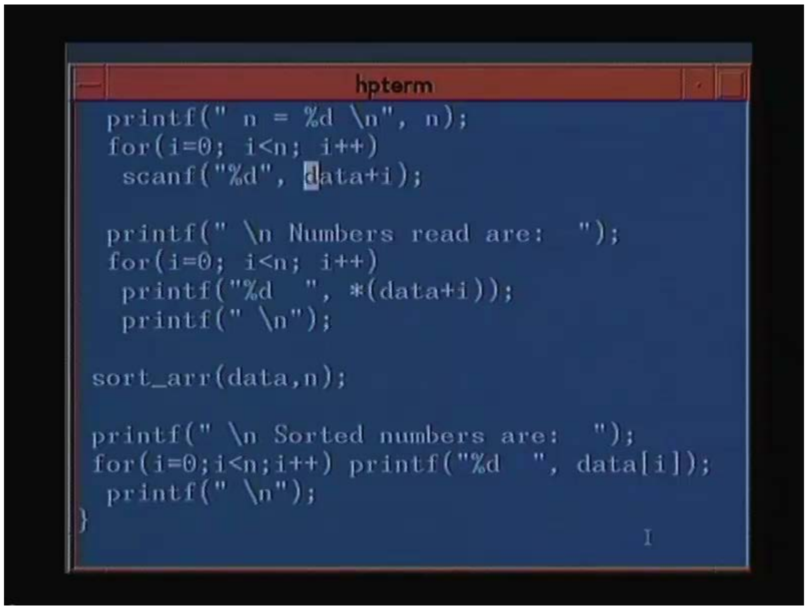
(Refer Slide Time: 22:00)



```
  printf(" \n Numbers read are:  ");
  for(i=0; i<n; i++)
   printf("%d  ", *(data+i));
   printf(" \n");

"arr1.c" 28 lines, 527 characters
$ cc arr1.c
$ ./a.out
Give n :5
 n = 5
1 4 3 2 5

 Numbers read are:  1  4  3  2  5

 Sorted numbers are:  5  4  3  2  1
$ vi arr2.
```

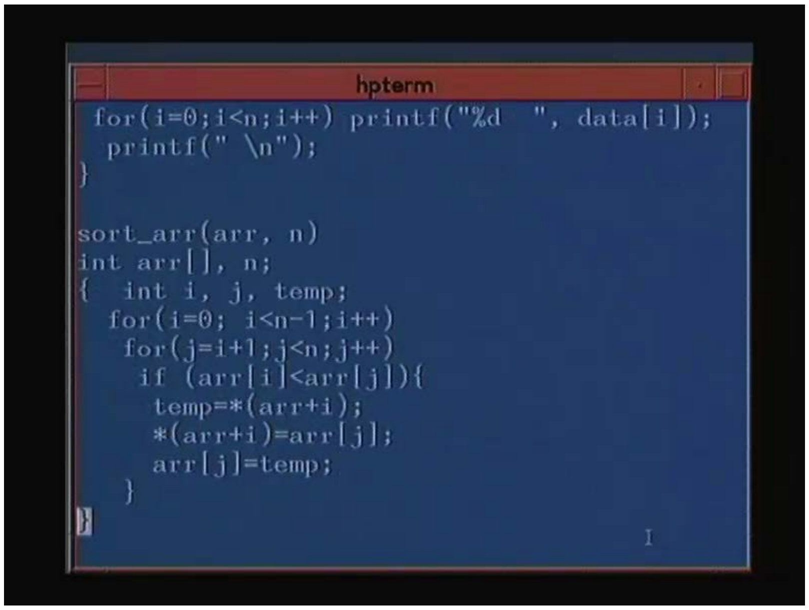
Coming to the second example, here we use a function and this function is called sort array. Here again we have read data plus i, initially printed star of data plus i and called sort_arr (data, n).

(Refer Slide Time: 22:11)


```
 printf(" n = %d \n", n);
 for(i=0; i<n; i++)
  scanf("%d", data+i);

 printf(" \n Numbers read are:  ");
 for(i=0; i<n; i++)
  printf("%d  ", *(data+i));
  printf(" \n");

sort_arr(data,n);

printf(" \n Sorted numbers are:  ");
for(i=0;i<n;i++) printf("%d  ", data[i]);
 printf(" \n");
}
```

And after sorting we printed the sorted. Now in the sorting as we did before int arr this, we don't have to give the value because we don't allocate any element, we have only own the address.

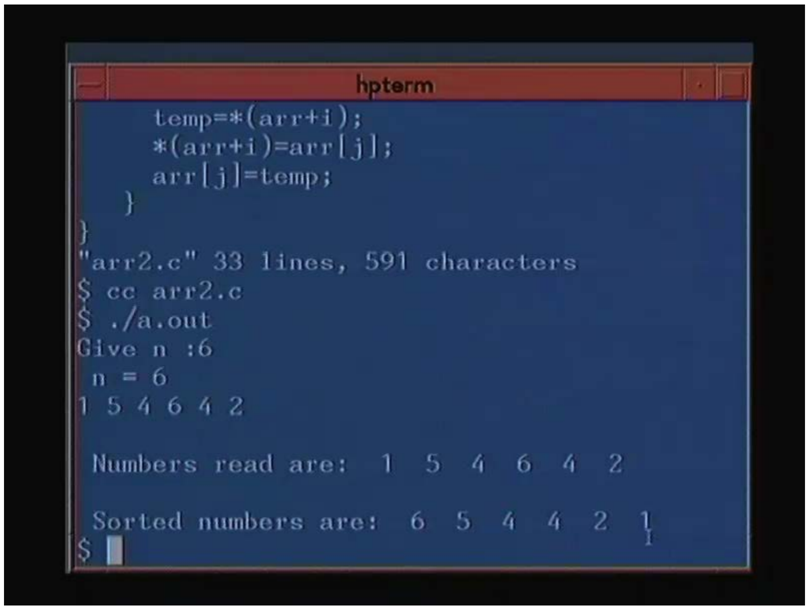
(Refer Slide Time: 22:55)


```
 for(i=0;i<n;i++) printf("%d  ", data[i]);
  printf(" \n");
}

sort_arr(arr, n)
int arr[], n;
{  int i, j, temp;
  for(i=0; i<n-1;i++)
   for(j=i+1;j<n;j++)
    if (arr[i]<arr[j]){
     temp=*(arr+i);
     *(arr+i)=arr[j];
     arr[j]=temp;
   }
}
```

Though we cannot change this address we will utilize this address and we can similarly use the star of arr plus i, we can use the content and access it by the array.

We can access it both ways, see we can access it as an array like this or we can access it like this, both accesses are allowed.

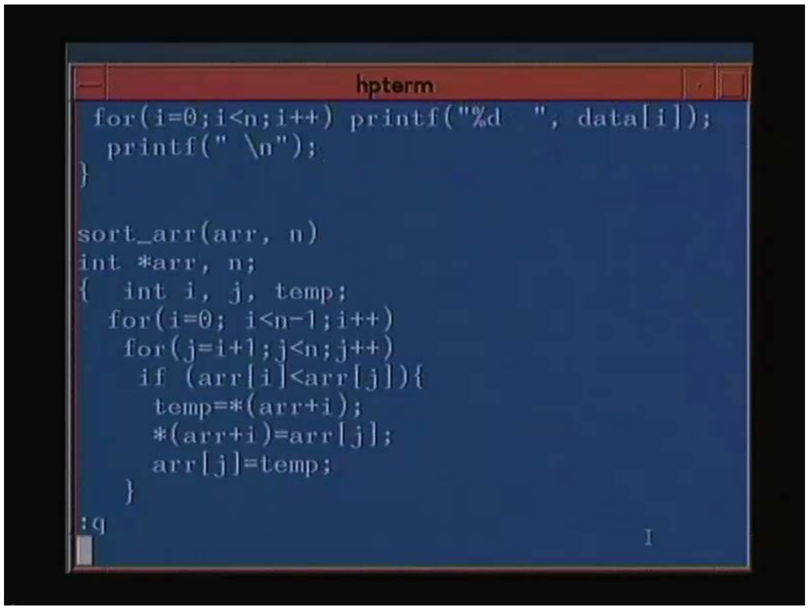
(Refer Slide Time: 23:12)



```
      temp=*(arr+i);
      *(arr+i)=arr[j];
      arr[j]=temp;
    }
}
"arr2.c" 33 lines, 591 characters
$ cc arr2.c
$ ./a.out
Give n :6
 n = 6
1 5 4 6 4 2

 Numbers read are:  1  5  4  6  4  2

 Sorted numbers are:  6  5  4  4  2  1
$ 
```

This way the same thing can be done. Let us see the third example. The third example is identical to the second example except this declaration. Here we declared only a pointer because that's all we need, we don't need to allocate elements.

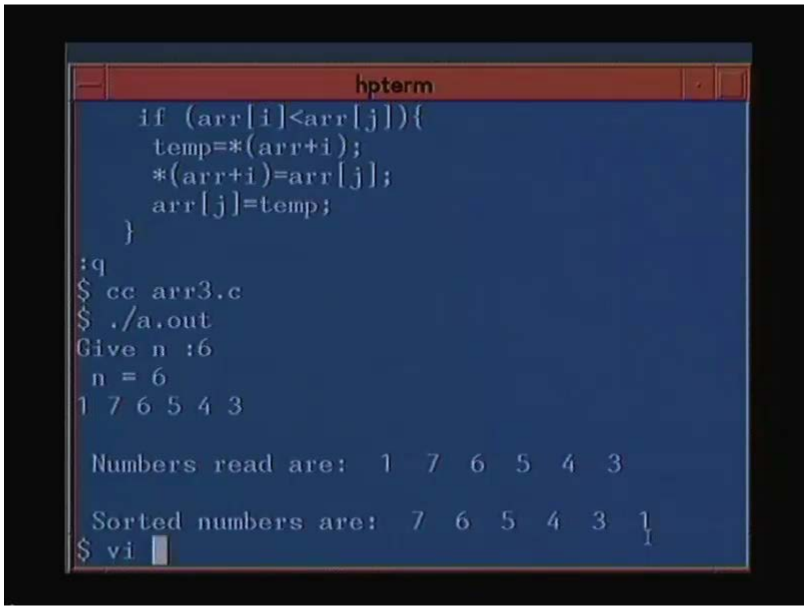
(Refer Slide Time: 24:15)



```
 for(i=0;i<n;i++) printf("%d  ", data[i]);
  printf(" \n");
}

sort_arr(arr, n)
int *arr, n;
{  int i, j, temp;
  for(i=0; i<n-1;i++)
   for(j=i+1;j<n;j++)
    if (arr[i]<arr[j]){
      temp=*(arr+i);
      *(arr+i)=arr[j];
      arr[j]=temp;
    }
:q
```

In the previous case also we did not declare anything, we did not allocate anything. We just allocated, we just declared that the array address is a pointer to an integer. That is the content of arr which is equivalent to the start address of the array. What is this? This is AND arr 0 and what did we say? Arr, the content of arr is an integer, so the content of arr, AND arr zero is an integer which is quite okay and we use it identically and it will again cause no problems. So this is another way, this is called pointer to an integer. So it’s fine, now we will see given an array declaration what are the array indices, how are they allocated, can we print out their addresses and see how they are allocated.

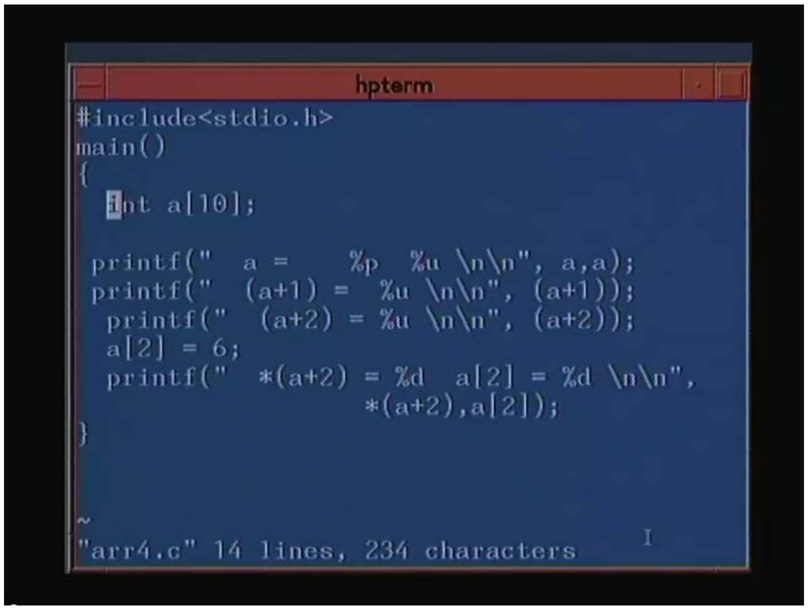
(Refer Slide Time: 24:30)



```
    if (arr[i]<arr[j]){
      temp=*(arr+i);
      *(arr+i)=arr[j];
      arr[j]=temp;
    }
:q
$ cc arr3.c
$ ./a.out
Give n :6
 n = 6
1 7 6 5 4 3

 Numbers read are:  1  7  6  5  4  3

 Sorted numbers are:  7  6  5  4  3  1
$ vi
```

So we declared a simple array of 10 elements 0 to 9 and we print A that is we print the address and to print the address, there is a percentage p that is type pointer. This is type pointer, it will give you sorry it will give you the value in hexa decimal format.

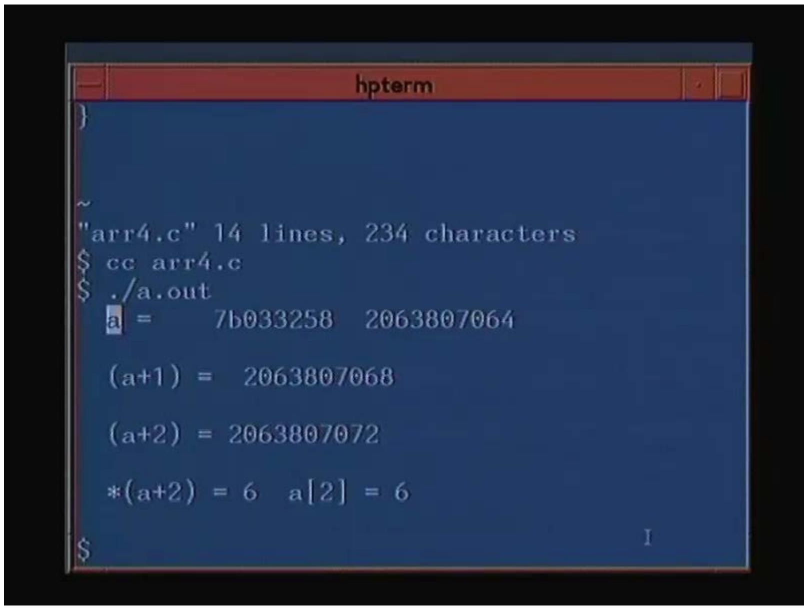
(Refer Slide Time: 24:50)



```
#include<stdio.h>
main()
{
  int a[10];

 printf("  a =    %p  %u \n\n", a,a);
 printf("  (a+1) =  %u \n\n", (a+1));
  printf("  (a+2) = %u \n\n", (a+2));
  a[2] = 6;
  printf("  *(a+2) = %d  a[2] = %d \n\n",
                  *(a+2),a[2]);
}

~
"arr4.c" 14 lines, 234 characters
```

And if you wanted in its integers format then you have to print it in unsigned. So it is declared as an unsigned integer. A pointer is an unsigned integer, it cannot take negative values, it can take only positive values that is why it is an unsigned integer. So % u was an unsigned integer. So first we printed a then we printed a plus 1 then we printed a plus 2 then we assigned a[2] to 6 and we printed star of a plus 2 and a[2] just to see and you will also be able to see from the difference between a and a plus 1 and a plus 1 and a plus 2 what is the number of bytes allocated for an integer.

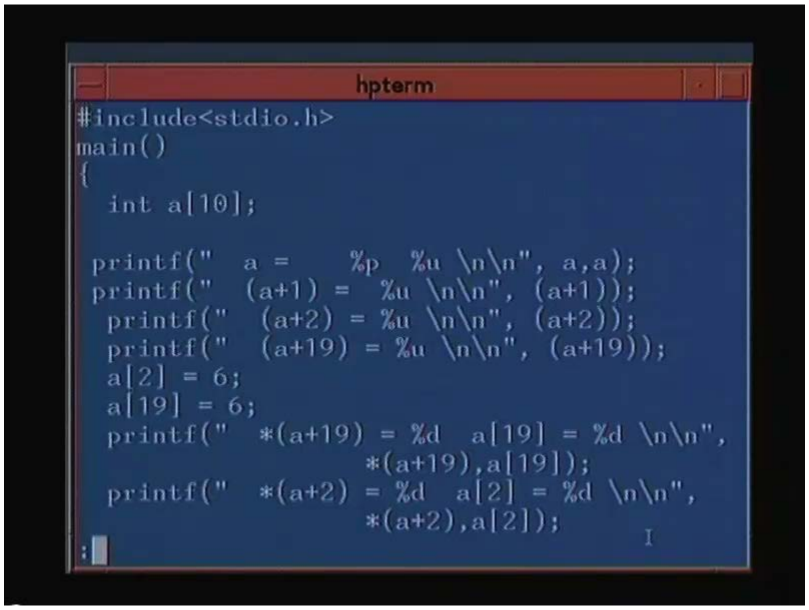
(Refer Slide Time: 26:18)



```
}


~
"arr4.c" 14 lines, 234 characters
$ cc arr4.c
$ ./a.out
  a =    7b033258  2063807064

  (a+1) =  2063807068

  (a+2) = 2063807072

  *(a+2) = 6  a[2] = 6

$
```

So a, the system allocated it to the location 7b033258 which is the hexa decimal value, its equivalent decimal value is the location 2063807064. A 1 is to adjustment the unsigned integer value here, 2063807068, so 4 places, 4 bytes are stored for an integer. This is 7 2 and then content of a plus 2 is 6 which we assume. Interestingly nobody will disturb you if you write down something like this. Possibly let's check.

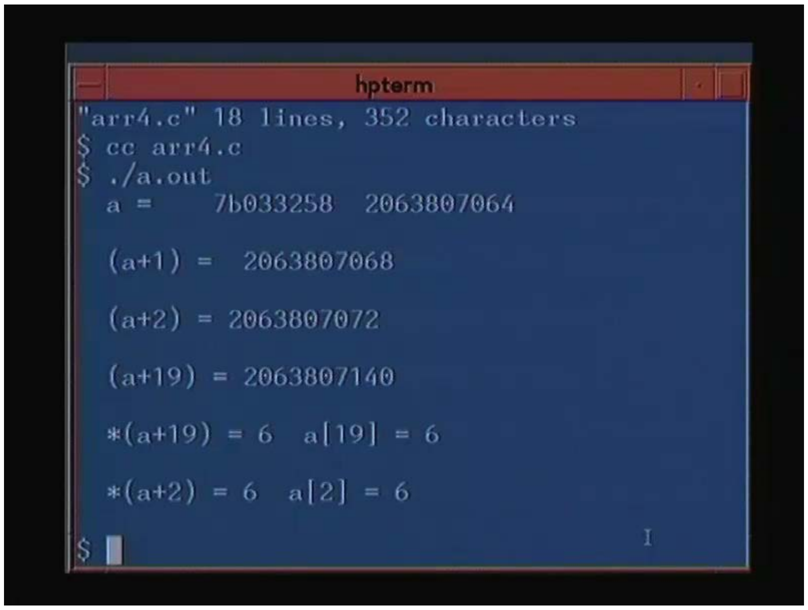
(Refer Slide Time: 28:21)



```
#include<stdio.h>
main()
{
  int a[10];

 printf("  a =    %p  %u \n\n", a,a);
 printf("  (a+1) =  %u \n\n", (a+1));
  printf("  (a+2) = %u \n\n", (a+2));
  printf("  (a+19) = %u \n\n", (a+19));
  a[2] = 6;
  a[19] = 6;
  printf("  *(a+19) = %d  a[19] = %d \n\n",
                   *(a+19),a[19]);
  printf("  *(a+2) = %d  a[2] = %d \n\n",
                   *(a+2),a[2]);
:
```

There is nothing wrong in writing this know because you are just writing down, a is value here adding it, it's an arithmetic expression so you will get the value. There is nothing wrong in this but the question is if you did this what will happen. You know what will happen, this is what will happen. So you have to be very careful about how you write a program.

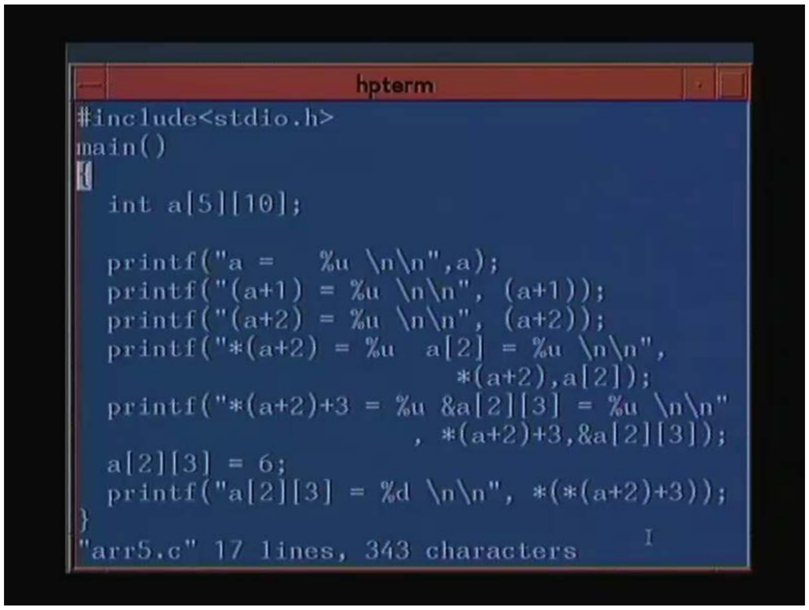
(Refer Slide Time: 28:35)



```
"arr4.c" 18 lines, 352 characters
$ cc arr4.c
$ ./a.out
  a =    7b033258  2063807064

  (a+1) =  2063807068

  (a+2) = 2063807072

  (a+19) = 2063807140

  *(a+19) = 6  a[19] = 6

  *(a+2) = 6  a[2] = 6

$ 
```

It will access that location and unless it is a location protected by the system, unless it is a system location that is location where the os is there or protected, it will simply go and write it. This is not pascal which will check their value out of range and array out of range and all that. In C nobody tells you what will happen, so please be very sure and very careful about what will happen because here this is computed, this address is computed and you are just saying this content of this address is equal to 6, it will be made 6. There is no automatic check but there are some other software through which if you run it, you will able to find out whether it does it.

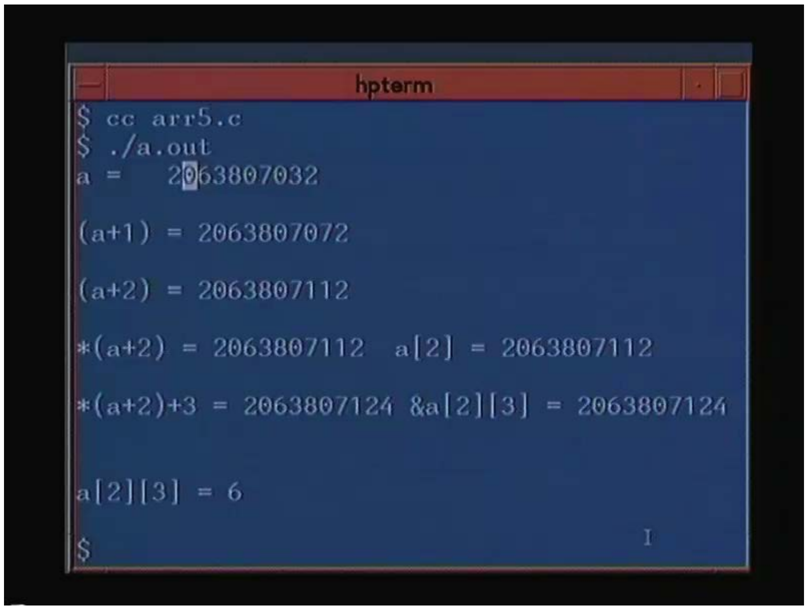
(Refer Slide Time: 29:45)



```
#include<stdio.h>
main()
{
  int a[5][10];

  printf("a =   %u \n\n",a);
  printf("(a+1) = %u \n\n", (a+1));
  printf("(a+2) = %u \n\n", (a+2));
  printf("*(a+2) = %u  a[2] = %u \n\n",
                              *(a+2),a[2]);
  printf("*(a+2)+3 = %u &a[2][3] = %u \n\n"
                          , *(a+2)+3,&a[2][3]);
  a[2][3] = 6;
  printf("a[2][3] = %d \n\n", *(*(a+2)+3));
}
"arr5.c" 17 lines, 343 characters
```

So now let us see the last example for a two dimensional array. Please, pay attention. Suppose we have a two dimensional array a, suppose we have a two dimensional array a [5] [10]. As we discussed there are 0 to 5, a 0 to a 5 is an array where each element is a one dimensional array of 10 integers that is how we can look at it. So a, this is the address a, a plus 1 a plus 2, content of a plus 2 will be same as a 2, content of a plus 2 plus 3 will be same as AND a [2][3]. A will be the same and AND a is 0 0 which will be the same as AND a 0. All of them will be same, AND a 0 is same as AND a [0] [0] is the same as a. a [2] [3] is the same as we were doing a bit earlier, content of the content of a plus 2 plus 3.

(Refer Slide Time: 31:15)



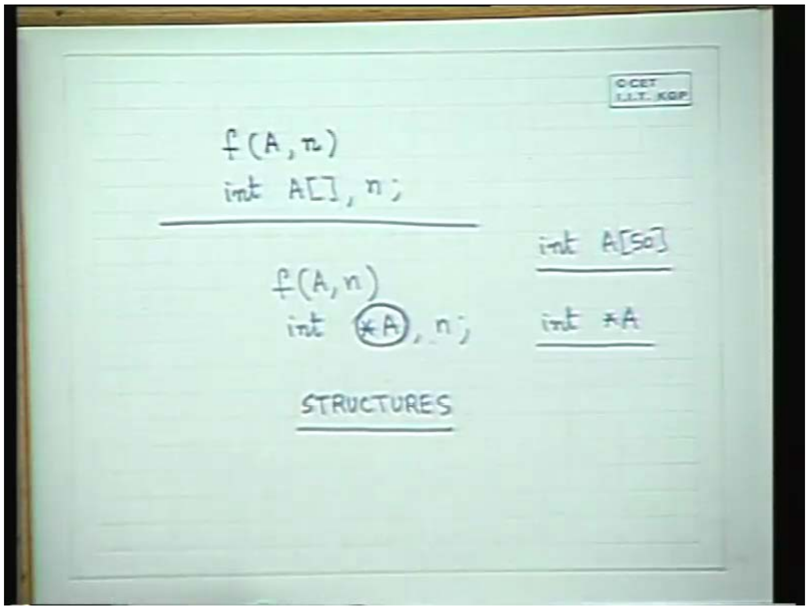
```
$ cc arr5.c
$ ./a.out
a =    2063807032

(a+1) = 2063807072

(a+2) = 2063807112

*(a+2) = 2063807112  a[2] = 2063807112

*(a+2)+3 = 2063807124 &a[2][3] = 2063807124


a[2][3] = 6

$
```

Now let us see. This is 2063807032 which is the start of the array, a plus 1 the difference is how many bytes, 40 bytes, you remember we declared it. So a plus 1 will give you the address of AND A [1] [0], similarly a plus 2 another 40 bytes. Content of a plus 2 which is the same as a plus 2 because a plus 2 is and a [2] [0]. Content of a plus 2 is a 2 which is the same as and a [2] [0]. So both will be, all this 3 will be identical, alright.

Now this one will be content of a plus 2 then plus 3 which will give you the location of the address AND a [2] [3] which will be 12 bytes after this, 4 into 3 12 bytes, so you have got 12 bytes now. And then a [2] [3] equal to 6, this one we can get the value of the location. Similarly you have to extend your ideas to 3 dimensions, 4 dimensions but there is a pattern and structure in the whole thing so it should not be a difficultly to understand, how to declare and use arrays what even part of arrays means something.

Part of an array is not an undefined quantity. Suppose you define a 10 dimensional array and you write only 5 of the dimensions then it also means something. So that is what has to be taken care of and what can be utilized during programming. So we will terminate with this discussion but we will remember 2 or 3 things. One is in the main program int A [50] and int star a are not identical. They are identical in the sense that this is an address to an array and this can be used be used as an address to an array but this does allocate 50 elements and this allocates only a pointer variable.

(Refer Slide Time: 34:10)



©CET
I.I.T. KGP

```
f(A, n)
int A[], n;
```

```
f(A, n)
  int *A, n;
```

```
int A[50]
```

```
int *A
```

STRUCTURES

This is a pointer variable, this variable can be changed. You can use this value and start changing it but here this array A cannot be changed. This is the address of the array which the system will allocate. Here this can be changed and then if you want to allocate 50, 30, 40, 60 elements then you will have to do dynamic allocation. And we will come to dynamic allocation a bit later on but before that we will move on to structures. So next day we will study structures in C.