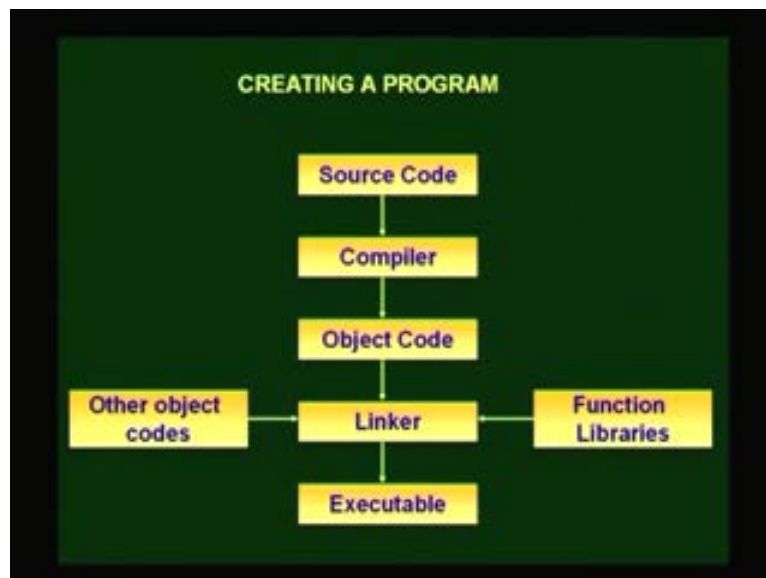


**Numerical Methods and Programming**  
**P. B. Sunil Kumar**  
**Department of Physics**  
**Indian Institute of Technology, Madras**  
**Lecture - 2**  
**Programing - Introduction to Pointers**

Hi. Welcome to the second lecture on numerical methods and programming. Today we will cover pointers and arrays. Then we will discuss how to use pointers and what is the special features, or **how it is, how** what are the differences between pointers and arrays; how do you use them interchangeably etcetera. So, before we do that, we will just go through what we discussed in the last class that is we talked about how to create a program, and the basic features of that. That is, what are the basic things which we have to do before we run a program?

Let us first create a source code and then you invoke a compiler to compile it, and **then use** that compiler creates an object code, and this object code is then linked to the other objects codes, or to a function library, using a linker. So now, this linking operation creates an executable, and that is what we would run on the computer. So this was one of the things we discussed in the last class.

(Refer Slide Time: 02:39)

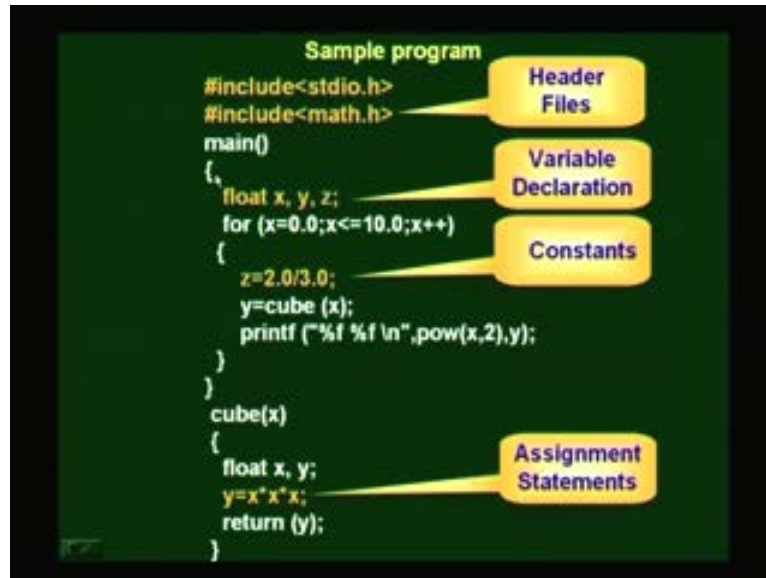


So we also discussed what are the general features of a program. So, here is the general features of the program. We have, beginning of the program, some header files okay like here, the standard input output header file, and the math .h header file, and then you have variable declarations, and then you have constants or other assignment statements. So there are the two examples of here: constant statements, or assignments statements. And I also said we discussed in detail what the different kinds of variables are we also said that when we invoke a compiler it is done in 4stages, that is, first there is the pre compiler which looks at all the "include" files right and then the second stage it looks at all variable declarations. All variables are

declared, etcetera. And in the third stage it creates the code, code understandable to the machine. And in the fourth stage, which if invoked, actually it rewrites the code to improve the performance. That is the optimization. That is the four stages of the code. That is what, that is the second aspect which we had discussed in the last class.

(Refer Slide Time: 4:14)

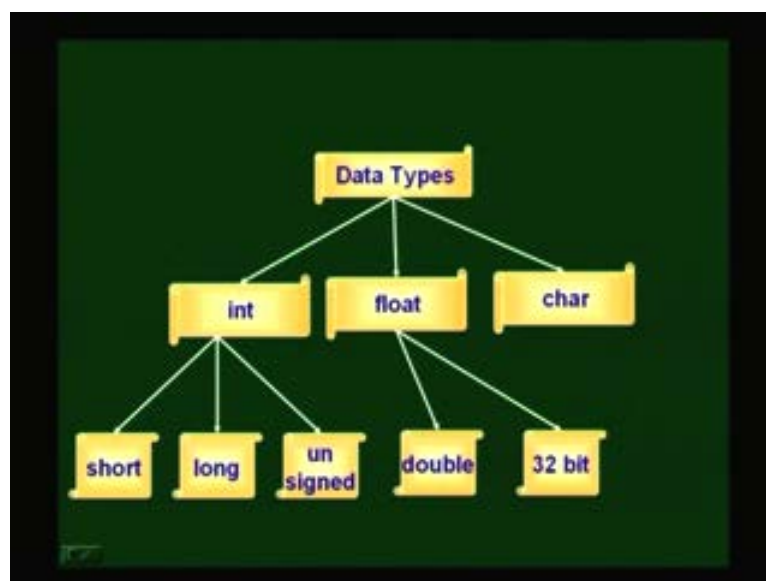
```
Sample program
#include<stdio.h>
#include<math.h>
main()
{
float x, y, z;
for (x=0.0;x<=10.0;x++)
{
z=2.0/3.0;
y=cube (x);
printf ("%f %f \n",pow(x,2),y);
}
}
cube(x)
{
float x, y;
y=x*x*x;
return (y);
}
```



The diagram shows a C program with four callout boxes pointing to specific parts of the code: 'Header Files' points to the #include statements, 'Variable Declaration' points to the float x, y, z; line, 'Constants' points to the z=2.0/3.0; line, and 'Assignment Statements' points to the y=x\*x\*x; line in the cube function.

And then we also discussed various data types, variable declarations, which we make; what are the various data types. We said we have integer types, we have floating points, and we have characters. These are the 4 different, 3 different data types which we discussed. Inside integers, we have short integers and long integers, and we have unsigned integers. So the biggest integer which you can get is the unsigned one because it even uses that bit which is reserved for the sign.

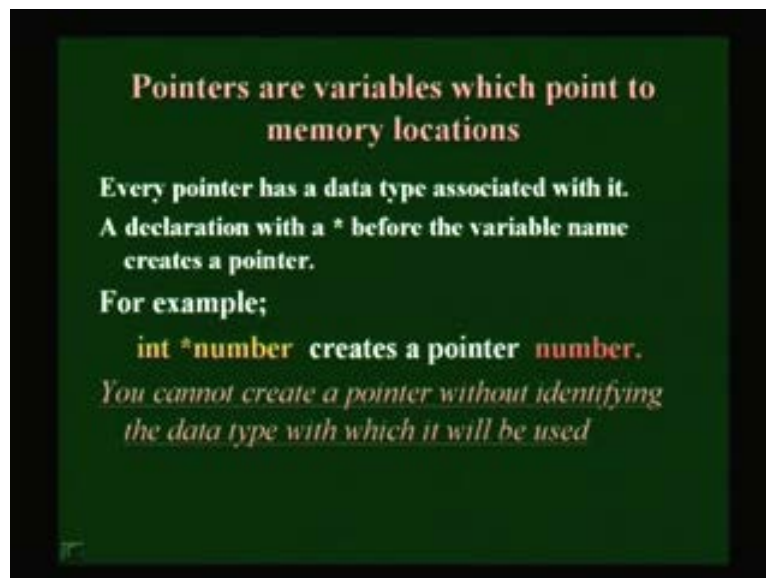
(Refer Slide Time: 05:15)



I also said that normally all compilers, when you say integer, it takes it as long integer, and floating point also you could have short, or single precision, or double precision. That is the different types of floating points. So, that is the different data types which we have. That is the summary of the last class. Today we will be discussing pointers and arrays okay. So what are pointers? Pointers are variables which point to memory locations, so it is something that is points to a memory location. We will see that how it is actually done in a little while. So like the variable, every pointer also has a data type associated with it. This is very important. Every pointer has to be associated with a data type. You cannot change data type of the pointer within a program. So how do we declare? So now we saw how to declare variables. Similarly, how do we declare a pointer? Very very similar as that of the variables.

The pointer is declared with an asterisk in front of the variable name. If you have an asterisk in front of a variable name, then that is a pointer. For example, so here is one of such declaration. Integer, and a star, an asterisk number. So, this creates a pointer called number. So, if you do not have this, then we know that this is a variable declaration for a number, an integer number. So, we have an asterisk here and then that is a pointer. So then, one important thing which you should remember is that you cannot create a pointer. I am repeating this: you cannot create a pointer without identifying the data type. So, that is very, very important.

(Refer Slide Time: 06:59)



First, you have to identify the data type, and then you create a pointer. We will see some examples of this here. So here is the example of the use of pointer. Actually there are two types of pointers. There is a file pointer, and there is a floating pointer, z here, is a floating point which is a pointer. If I said “float z”, it would have been a variable z, but I have a floating point pointer here. So this is variable z. Now I have, will show you how to use that. It will go to what actually happens in the code in a little while okay. So here I am using it. I am saying that, for example, normal floating point is x and y are normal floating point variables.

(Refer Slide Time: 7:20)

```
#include<math.h>
#include<stdio.h>

main()
{
float x,y;
float *z;
FILE *FP;
FP=fopen ("sample.dat","w");
y=10.0;
z=&y;
x=*z;
printf("%f %f\n",y,x);
}
```

So I could just assign a constant saying y is equal to 10. That is a floating point. I am assigning a value to that y, but z is a pointer. I cannot assign a value like that. How do we do this? That is what I wanted to show you here. So, for every variable, when you declare there is a name which we give to it, and then there is an address, and then there is some content in that address which we put. That is what we do okay. So when we say integer n, for example okay, so there is a name, n, which is given to it, and there is some address, some address to the memory location correct, and then there is the content. The content is here 0.

(Refer Slide Time: 09:40)

	Name	Address	Content
int n;	n	16254	0
int m;	m	16256	0
int *num;	*num	16258	0

We have not put in anything. So the content is nil and then I set integer m. Again it creates a name and an address and a content. Similarly, if I had set here floating point x and floating point y, so it would have a name and an address, and some content in that address, which whatever we put in it. Now let us say I declare a pointer, integer

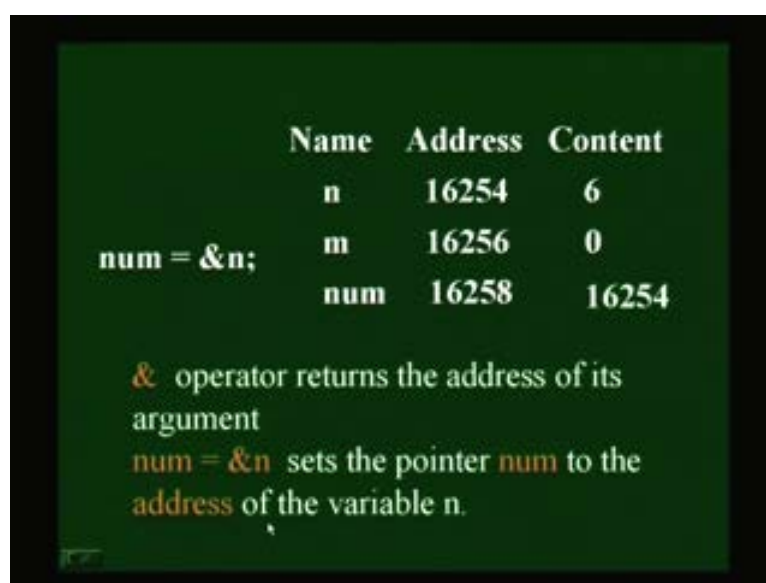
“star num”. Num is a pointer okay. So then what happens? Again, the name is now, the variable name is “star num” that is the variable name. It has an address and the content is 0. So now this is our initial thing. So now, let us say I do something like this. Here I have said y is equal to 10, right.

Similarly, in this example here I use an integer. Here I am using a floating point, and here I am using an integer. So when I say n is equal to 6, what happens? The content of that variable, n, changes to 6. So now that is what we did. We changed n equal to 6 and then now we are going to use, we are going to point the pointer to a location. So we have this pointer “num”. We want to now point that to an address of the variable, n. So we are going to point “num”, the pointer “num” to the variable n. So this ampersand sign, this is one of the operators associated with the pointer, there are two operators, one is the asterisk sign which we saw, and another is an ampersand sign.

So now this ampersand sign will actually return the address of the variable. So that is what this ampersand sign do. So if you have ampersand of any variable it returns the address of that variable and that address is now associated to this pointer. So what we see is that this pointer “num” notice the difference, here it is just “num” not “star num” okay the “num”, as a content is basically the address of n. So that is what we are doing. So here, for example, I said y equal to 10, and I said z is equal to ampersand y. Okay so this z is now pointing to the value pointing to the address y.

So when I say x is equal to star asterisk sign, z okay, it returns the value to which the pointer z is pointing. So where is z pointing ? z is pointing to y okay. Asterisk z star z will return the value of y, okay. So that is what we would see here again. So let us look at that what actually happens. The ampersand operator actually returns the address of its argument, remember that. That is one argument which we use, one operator which we use is the ampersand operator. Now, so I am just summarizing this again. So you say “num” equal to ampersand n means it sets the pointer “num” to the address of the variable n. So that is one of the basic usages of that.

(Refer Slide Time:12:38)



	Name	Address	Content
	n	16254	6
num = &n;	m	16256	0
	num	16258	16254

& operator returns the address of its argument  
num = &n sets the pointer num to the address of the variable n.

Now let us see. We go ahead and see what happens if you assign something like x is equal to star z which I said there. So here I say m equal to star num. Remember, num is pointing to variable n. So what should happen to m? So m should get the value which is that of n, because this star num will return the value of the variable to which num was pointing right. So we would make that m to go to 6. So other one the star num had the value of n. That is what this is. That is what we said. We saw another operator, that is called the indirection operator right. What we saw was ampersand operator which actually points to a particular address okay, and then we have this indirection operator okay, which is the asterisk sign, which sets the value of m to that pointed by num okay. So here it sets the value of x to that pointed by z, okay.

(Refer Slide time: 14:18)

```
Sample.c

#include<math.h>
#include<stdio.h>
main()
{
    float x,y;
    float *z;
    FILE *FP;
    FP=fopen("sample.dat","w");
```

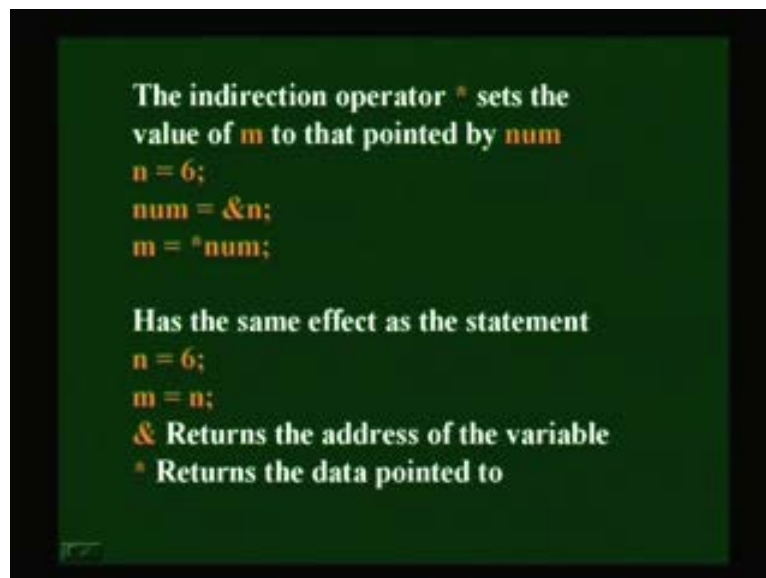
(Refer Slide Time: 14:33)

```
    y=10.0;
    z=&y;
    x=*z;
    printf("%f %f\n",y,*z);
}
```

If you want to print out the value to which z is pointing, then you would again use here, for example, you can use here star z instead of x. So we will see how this will work. We will just execute the program and we will see what we will get. Okay, it is printing out two numbers, 10. Let us say y equal to 10, I assigned. I pointed z equal to ampersand y, pointed z to y and then I assigned to x the value to which z is pointing, so that is 10. So both of them will get set. I could also do this. I could just also, will try this, and it prints the same thing okay.

So, it basically returns the value to which it is pointing to, okay. So now, for example, let us make this statement n equal to 6, and we say num equal to ampersand n, and then m equal to star num right. That is same as, that has the same effect as the statement n equal to 6, and m equal to n. So why do you want to use this, why do you want to use this ampersand sign and pointer? We will see that in a little while, okay.

(Refer Slide Time: 16:22)



So the summary of this part is that we had the ampersand sign to return the address, and we have the asterisk sign to return the, to give the, to return the data to which it is pointing. So that is the two basic operators associated with pointers. Okay so now we saw now how to use pointers on one side. For example, in this program we use pointers on this side using an asterisk sign, and pointers on this side where we have to use an ampersand sign. So this is not necessary. It is not the only way to do it. We can have pointers on either side.

So that is what we would like to see. So let us again look at that. So we had n equal to 6, and n pointed to that right, and then we said z equal to ampersand n, and then z actually pointed to that address right, okay. Now we say m equal to star z, so that m would take the value of n. So what we can do is now I can say this asterisk z is equal to 9. That is using the pointer on the other side right. Here, I had used star z on the right hand side of this equation. We can also use it on the left hand side of the equation. That is what this is. So here that is what we are doing. We are using the other side.

So now, what would happen is that, see my pointer z is pointed to n, to address n right, and I am changing, when I say asterisk z which basically is returning the content of the variable to which z is pointing, which is n. So now, if I do that okay so then I will actually set the asterisk z to 9, and then that would change the value of n to 9, but the value of m will not change. So that is that is the difference here. So if I change this because I set z to point to the address of the variable n right, if I change asterisk z, the value of that, that is the value to which z is pointing, that will change the value of n but m is given a constant assignment there. That does not change. So that is the difference between assigning a constant and pointing it to a and using a pointer.

(Refer Slide Time: 19:38)

**Pointers can be used on both sides of the assignment equation**

	Name	Address	Content
n = 6;	n	16254	9
z = &n;	z	16256	16254
m = *z;	m	16256	6
*z = 9;	*z	16256	9

(Refer Slide Time: 20:00)

```

#include <math.h>
#include <stdio.h>

main()
{
float x,y;
float *z;

y=10.0;
z=&y;
x=*z;
printf("Nf Nf\n",y,x);

*z=9.0;
printf("Nf Nf\n",y,x);
}
"samp11.c" 17L, 197C

```

So that gives you one of the differences. So this screen is basically supposed to illustrate two things: (1) I can use this on the left hand side, not necessarily only on



the right hand side, and that if I change if I point a pointer to a variable, and if I change the value of the pointer, that is the data to which it is pointing, then that variable also automatically takes that value. That is what this is. We can look at that using an example again. So here I am trying to show exactly the same, with the example, I am trying to demonstrate what is listed here.

Okay, so here I have a program in which I have declared a pointer z again a floating point pointer, and I have two floating points x and y. So now what do I do? I have given a constant assignment here, y equal to 10, and I just going to point my pointer to that variable y right okay and then I am going to assign that data which the pointer z is pointing to, to variable x right, and then I will print out that here. So I will print out both y, which will be 10 here, and x, which should be 10 because x is given a value to which z is pointing which is 10, and then I will change star z to 9 okay and then I want to print out y and x again. That is what we are going to do.

Okay so we have the first printout, that is, when we print it here, I got y as 10 and x as 10, and then I change the data to which z is pointing okay. That is basically the value of y. I changed it to 9, but that does not change x. So when I changed this one it does not change the x, it changes the y okay. That is what we will see here. So it changes the y to 9, but x remains 10. That is what I have said here, basically, that is exactly what I have said here. I put n to 6 and I point z to n, and then I assign to m the value which z is pointing to, the variable to which z is pointing, and then I change that variable, that is star z, I change to 9, then m does not change, but n will change.

That is what we see here, because z is just basically pointing to this address. So whatever you do to z, it will change that, but it will not change this value because here, which is given m a content and not just an address. So that is exactly what we see here now okay. So here we just gave z to point to y, and I took the value of y and gave it to x, and then I changed z, but z is actually a pointer, remember.

(Refer Slide Time: 23:16)

```
sample1.c
#include<math.h>
#include<stdio.h>
main()
{
    float x,y;
    float *z;
    y=10.0;
    z=&y;
    x=*z;
```

(Refer Slide Time: 23:31)

```
printf("%f %f\n",y,x);
*z=9.0;
printf("%f %f\n",y,x);
}
```

So I change the value of z, means it will change the variable to which it was pointing. That is, it will change y. So y will change x will remain the same, and that is what we saw here, and I printed it twice. First time, both were 10, and then the y changed, and x remained the same. So that is the summary; the changing asterisk z changes n, but not m okay. So now we look at the next one. That is the array. So we discussed pointers, now we will look at arrays. So here is an example of that. I am declaring an array, x integer type. Again, arrays also have to have a data type just like pointers and other variables. Arrays also should have the data type, so here is an integer data type array. X, it is an integer data type, so 4 is the dimension of the array. That means, you can put four different values into this array.

(Refer Slide Time: 23:56)

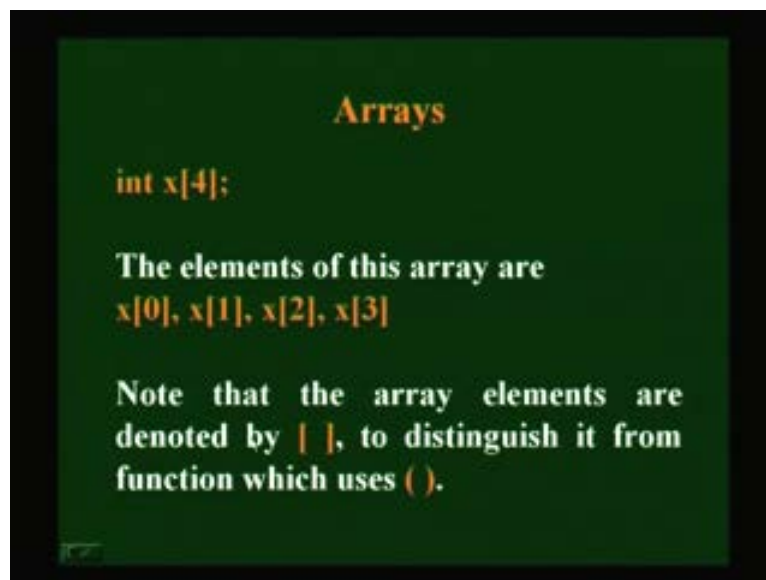
**Pointers can be used on both sides of the assignment equation**

	Name	Address	Content
n = 6;	n	16254	9
z = &n;	z	16256	16254
m = *z;	m	16256	6
*z = 9;	*z	16256	9

**Note that changing \*z changes n but not m**

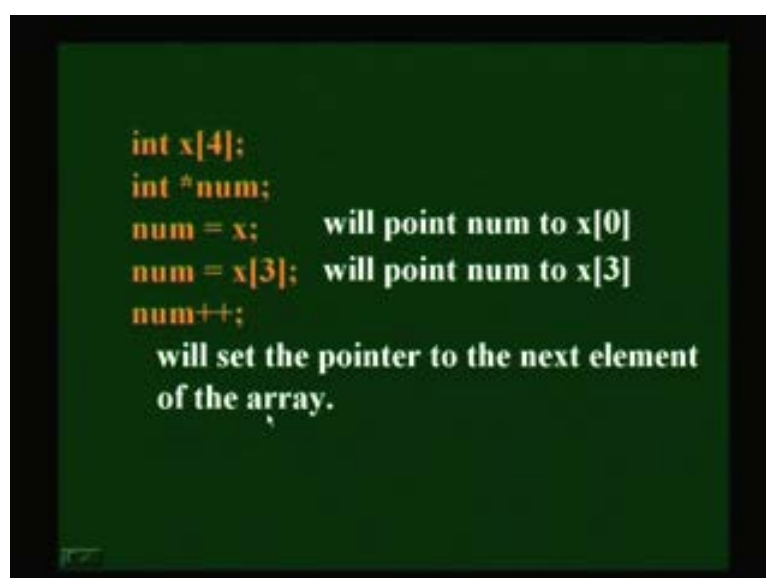
We will see that. So the elements, there are 4 different elements in this array. The elements of this array are x 0, x 1, x 2, and x 3. Remember, it starts from 0 okay. That is special. Remember this always. When your dimension of the array is 4 it basically ends in 3, because it starts from 0. The counting starts from 0. Okay so now, we use a square bracket instead of a bracket, like this, so just to distinguish it between a function and an array. So instead of a simple bracket, we use a square bracket.

(Refer Slide Time: 25:27)



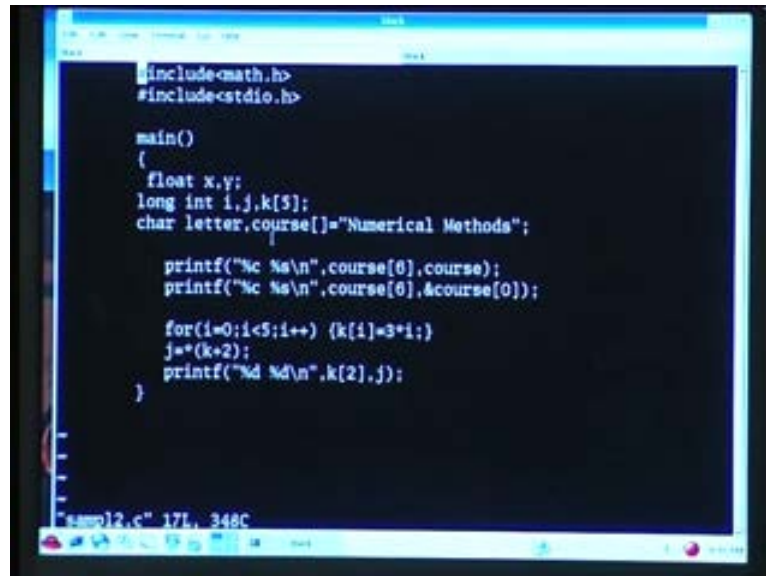
Now we would see some uses of it, and then we would have a more clear idea of what it is. Here is a small program which says declares an array and a pointer, and then I am saying that num equal to x. So now, this is an interesting point here. Num equal to x, if I say, actually num points to the first element of the array. We could also say num equal to x of 0. It is the same, but if you want to point the pointer, num, to any other value, then you have to say x of 1, x of 2, etcetera.

(Refer Slide Time: 26:45)



Okay that is what we would see. Num equals x of 3, you have to say, will point to x of 3. Okay we will see some of the examples of this. So here is an example which uses both an integer array and a character array. Before we go into this thing, we will see some examples of this. So, num plus plus you can also do some arithmetic with it. We could say num plus plus, that will increase the pointer by 1. Okay so we will see examples of all this. So when we say plus plus, we would actually set the pointer to the next element of the array. So that is what we are going to see in an example.

(Refer Slide Time: 27:12)



```
#include<math.h>
#include<stdio.h>

main()
{
    float x,y;
    long int i,j,k[5];
    char letter,course[]="Numerical Methods";

    printf("%c %s\n",course[6],course);
    printf("%c %s\n",course[6],&course[0]);

    for(i=0;i<5;i++) {k[i]=3*i;}
    j=*(k+2);
    printf("%d %d\n",k[2],j);
}
```

Okay so here is an example okay which uses both an integer array okay and the character array. So character strings are actually arrays. If I say here, for example, I have declared “course” as an array okay, so it is a string numerical methods. So if I say, course 0, it will pick up n; course 1, it should pick up u. That is what we are trying to show here. Let us just execute this program and then let us just go through this first, and then we will see what we should do. So when you want to print a character you use “percentage c” okay. When you want to print a string you say “percentage s”. Here, when you say “percentage s course” okay now I said course is a string. So, if I say “course”, it will print the whole numerical method, while course 6 is an element of that array.

So it is one character. So it will print out the 6th character on this, that is, this is 0, this is 1, 2, 3, 4, 5, 6, so it will print out the 6th one. So that is what you have to see, and then similarly, I have trying to show you that if I say percentage course 0, ampersand course 0, it will print exactly the same thing. So these two mean the same thing because the course is also ... an array name is also a pointer to it. That is what we are trying to see here. Okay we will this here, and then there is the use of an integer array where I am initializing the integer array using an assignment statement here. So, I run the i loop, this loop from 0 to 5, and I am putting that into this array.

And then I can actually pull out the values from that array using this sign because k, which is the name of that array is also a pointer to that array. Those are the two things

which we want to see here in this program. So we will see that in a minute. Okay so, as I said here, there is some pointer arithmetic, num plus plus, and num plus 2, num plus plus sets it to the next element. If you want to go two elements ahead, you can say num plus 2. So it sets the pointer to the two elements ahead.

(Refer Slide Time: 29:36)

```
int x[4];
int *num;
num = x;    will point num to x[0]
num = x[3]; will point num to x[3]
num++;
    will set the pointer to the next element
    of the array.
num +2;
    will set the pointer to the next to next
    element etc.
```

So that is the integer array which we have used here. Here, I have an example of that here again. So I put in this value. The array name is also a pointer to the start of the array. So what I wanted to say in these two slides was, one is, that I can define an array using a statement like this, so this will have array elements starting from 0 to 9; and then I can assign this is not a complete statement, this is not a full program statement, this is just an indication that I am starting from i equal to 0, and I am going in steps of 1, i plus plus.

(Refer Slide time: 30:54)

```
int m;
int n[10];
for (i=0; i++)
{
    n[i] = 3i;
}

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

I should actually put in “a”, so that is what I wanted to say. I want to run this loop; I want to put up to 9; I can go up to 9 inside this okay. If I say now n, n is actually a pointer, just say n, then n is a pointer to that array. That is what I am doing here. I am saying that k is now a pointer to the array which is 5 elements here, 0 to 4 okay, so I can say star of k plus 2 okay. Note this bracket here, it is required. So, star of k plus 2, that will take the 2 nd element of this array and return its value and give it to j. That is what we will see. So we will just print out this. We will just run this program and then see what we will get okay.

So let us look there what we wanted to print. We wanted to print it using two different ways. I said course 6, that is the 6th element of the array. So that turned out to be c right, and that is 1, 2, 3, 4, 5 okay that is starting from 0, 1, 2, 3, 4, 5, 6. That is the 6th element of the array, 7 th element of the array course of 6. That is the 7th element starting from 0 and then I am printing out the course which is the string, so I get here element is 6, but the string is numerical methods, and I am printing the same thing in a slightly different way. Again, I am printing the 6th element here, the 7th element, and I am using ampersand course 0. It is actually pointing to course because course is a pointer okay, to the array. So I can get the same thing by doing that.

So all array names are pointers to the array, to the first element of the array, but it is a constant pointer. You cannot change it. So that is a constant pointer. There is a difference between this and the other pointer. So this is a constant pointer. You cannot change that value. It always point to the, it always points to the first element of the array. So that is the difference okay. So here again I had this similar to what I have shown here, I had this k of i equal to 3 star i. Now I want to pick up different elements of this array. I can use k as the pointer for that okay, but k always points to the first element. So I have to use, if I want to point to other elements, for example, the third element, I have to say k plus 2 okay. So then that points to the 3rd element, that is k of 2. So that is what we printed out here and we got it as 6.

(Refer Slide Time: 34:07)

```
sample2.c

#include<math.h>
#include<stdio.h>
main()
{
    float x,y;
    long int i,j,k[5];
    char letter,course[]="Numerical
                          Methods";
```

Let us put, remember, if it is a normal pointer we could say asterisk k, star k would return the value to which k is pointing to, but since you want to point to the different elements of the array, you have to increment that pointer, but when you do that, that increment, since k is a constant pointer, you have to use this bracket. That is the important difference between a pointer declaration and an array pointer okay. So okay so now, so we will see that, summarize that, again. So m equal to n of 0, and m equal to star n okay, are the same thing okay. They have the same effect. That is what we just saw here. k of 2 and k plus 2 give the same value. So that is exactly what I demonstrated here to you, that if I say k plus 2, and I take the star of that, or I say k of 2, I get the same values. That is what we just demonstrated.

(Refer Slide Time: 34:22)

```
printf("%c %s\n",course[6],course);
printf("%c %s\n",course[6],&course[0]);
for(i=0;i<5;i++)
{
    k[i]=3*i;
}
j=*(k+2);
printf("%d %d\n",k[2],j);
}
```

(Refer Slide Time: 36:30)

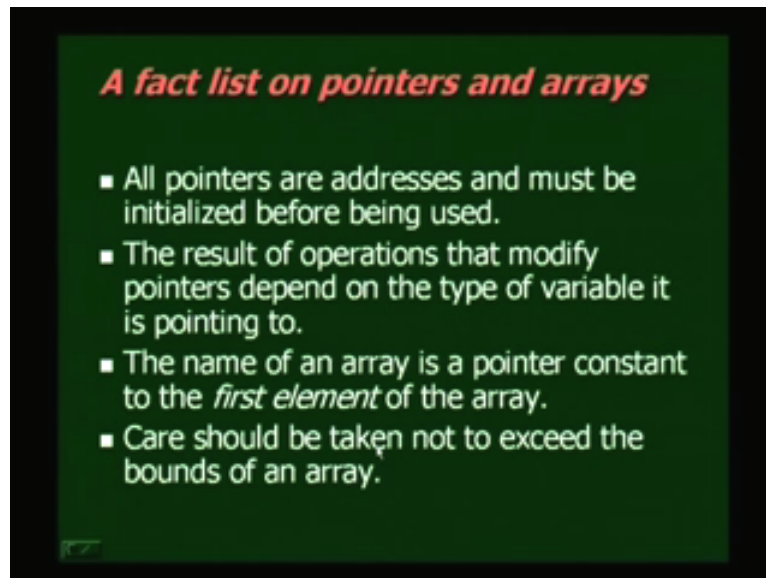
### Character Strings

**char course[] = "Numerical Methods"**  
**course** is a pointer to the character array.  
**course[0]** is initialized to "N"  
**course[1]** is initialized to "u"  
.....  
.....  
**course[16]** is initialized to "s"

Okay, so here is the summary of the character strings which I showed. So the course is again a pointer to that character array. Course 0 is now being initialized to n, and

course 1 to u, etcetera right. Then course 16 initialized to s, so that is what it is. That is what we just saw in this program. We did exactly the same thing here. So it is just a summary of what we just saw in the program. Okay so let me, before we go into the next thing, let me just give you a fact list on pointers and arrays okay. So all pointers are addresses and must be initialized before we use it okay. So the result of operations that modify pointers depends on the type of the variable to which it points to, by which I mean that how many, we have seen the address, when you say that “a” is a pointer, and when I say a plus plus, what does it mean?

(Refer Slide Time: 38:31)



(Refer Slide Time: 38:48)

```
include<math.h>
#include<stdio.h>

main()
{
    float x,y;
    long int i,j,m[6],*k[6];
    char *letter,*course[3],name[]="I I T Madras";

    course[0]="Numerical Methods";
    course[1]="and Programming";

    printf("%s %s\n",course[0],course[1]);

    for(i=0;i<5;i++) {m[i]=3*i;}
    k[0]=&m[1];
    k[3]=&m[4];
    j=k[3]-k[0];
    printf("%d %d %d\n",j,*k[0]+1,*k[3]);

    letter=course[0];
"samp13.c" 30L, 644C
```

So how many, what is the change of the address? Where it is pointing to will change depending upon whether a is an integer or a is a 32 bit floating point or a 64 bit floating point, etcetera correct. That is what the meaning of the statement is. The result of operations that modify pointers will depend on the type of variable it is



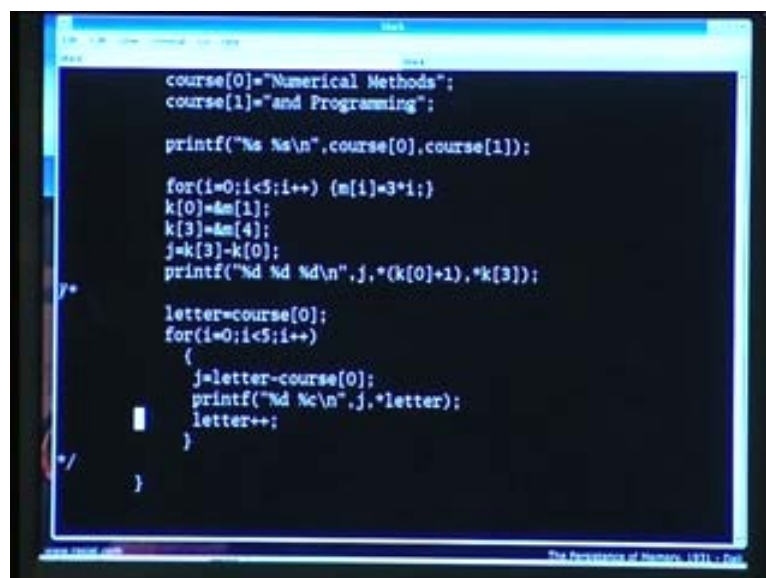
pointing to. Okay, the name of an array is a pointer constant to the first element of the array. You saw that just now, and when you create an array and when you use it, care should be taken not to exceed the bounds of an array. This is a warning for all C program users, that C program, C compilers, do not check the bounds of an array. So, as I said here that when I say k of 5, it can only store 5 elements starting from 0 to 4.

If you try to put a 6th element with it the compiler does not check it okay. So you should take that care. So, this is the price you pay for some flexibility. So this can have disastrous consequences if you exceed the bounds of an array, because it can write to some other memory location and it can be disastrous okay. So now before we go into the concept of pointers to pointers, we will look at some more examples of pointer arithmetic to get more familiar with pointers. So here is another program which would demonstrate the use of pointers. Now this is a little longer one.

So now I have used here two different character pointers, that is, star letter, and a pointer array. Okay now this has 4 elements in that array, and I have used another character array. So there is a pointer array here which is a list of pointers; okay here is a single pointer; and then I have a character string; three different quantities. Similarly, I have an integer array, and a pointer, a set of pointers, and I just want to show in this program how do we use this. So I can say, for example, course, 0 course 1 course, 2 etcetera, are pointers. So I am using course 0 here to point to numerical methods okay, and I am using course 1 to point to programming, and programming, there are two strings which I am pointing to okay.

So now if I print that using this statement, that is, both are strings, I will get numerical methods and programming correct. So similarly, I have this array here, an array m, so I just ran it up to fill it up with this assignment statement m of i equal to 3 i. So m of 0 is 0, m of 1 is 3, m of 2 is 6, etcetera, and then I am pointing k of 0. Now, I have a string of pointers here, integer pointers, and I am pointing that to m of 1, and I am pointing k of 3 to m of 4. So these are pointers, and I can subtract two pointers. I can do a little bit of pointer arithmetic. This is just to demonstrate that.

(Refer Slide Time: 41:43)



```
course[0]="Numerical Methods";
course[1]="and Programming";

printf("Ns Ns\n",course[0],course[1]);

for(i=0;i<5;i++) {m[i]=3*i;}
k[0]=&m[1];
k[3]=&m[4];
j=k[3]-k[0];
printf("Nd Nd Nd\n",j,*k[0]+1,*k[3]);

letter=course[0];
for(i=0;i<5;i++)
{
j=letter-course[0];
printf("Nd Nc\n",j,*letter);
letter++;
}
}
```

So I can do  $k$  of 3 minus  $k$  of 0, and then I can get an integer value. Both should be careful when we use this kind of pointer arithmetic, that both pointers should be of the same type. You cannot subtract an integer pointer from a character pointer. Both have to be the same. If you have similar types of pointers and then we can use, we can subtract like this, and we will run that okay and then let us run it up to that, and I have commented the program beyond this okay, so that this is a comment statement which you might know. This is basically an instruction to the compiler that you ignore everything starting from this to this, so the program ends here. So the program basically stops by printing out the value  $j$  which is the difference between these two pointers okay, and it would print star  $k$  of 0 plus 1. So, I said  $k$  is also a pointer to the array, to the first element of the array, so it will print that okay, and then it will also print star  $k$  of 3.  $k$  of 3 is pointing to the 4th element of the array  $m$  of 4, the 5th element of the array.

(Refer Slide Time: 42:35)

```

bash-2.05b$ gcc sampl2.c
bash-2.05b$ ./a.out
c Numerical Methods
c Numerical Methods
6 6
bash-2.05b$ gcc sampl3.c
bash-2.05b$ ./a.out
Numerical Methods and Programming
3 6 12
bash-2.05b$

```

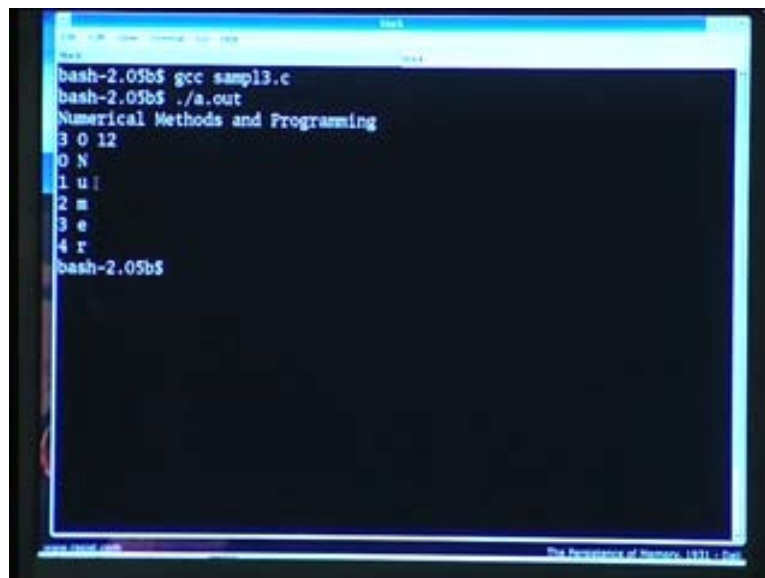
So let us compile this and then print it and then we will see. Okay so we have the numerical methods and programing printed, and we have 3, 6 and 12 printed. Let us go back and see what is this. So first, that when I subtract, okay  $k$  of 3 from  $k$  of 0, I get the difference between these two pointers. So one is pointing to the 0th element, the other is pointing to the 3rd element. So you get 3. You can do that only basically. You can get this, use this, to check the length of the string okay. That is the use of that okay. So if you want to know what the length of the string is you could actually take the pointer of the first and last elements and take the difference.

We will see the use of that later okay, and then this  $k$  of 0 plus 1 is pointing to the 2nd element, and  $k$  of 3 is pointing to the 3rd element. That is what we just saw,  $k$  of 3 is the 4th element, that is, 0, 1, 2, 3, sorry,  $k$  of 3 is pointing to the 5th element  $m$  of 4, that is what we have printed out here. So I have assigned  $k$  of 3 to the 5th element, that is, 1st element is 0, 2nd one is 3, and 3rd one is, that is,  $m$  of 2 is 6, and  $m$  of 3 should be 9, and  $m$  of 4 should be 12 okay. That is what we have printed out, okay.

So that is what it is pointing to while  $k$  of 0 is 0, and  $k$  of 0 plus 1. That is what we have printed here as 3, 6, I am sorry. So, this is the same as I will show you, this is the same as using, this and this would, I could use  $m$ , which is a pointer, or which is the array name as a pointer, and use that, or I could use  $k$  which is a pointer itself okay. So I could use, I could assign  $k$  of 0 or  $k$  of 3. I could use both of them. I could use  $k$  of 3 also which is another pointer okay. So it prints out 0 okay. So I could use here a pointer, or I could use the array name, one of them okay, in this thing, this program, basically demonstrates that you can subtract the same type of pointers that will give you the array element and that you can use either pointer itself here or you could use the array name okay, both are possible.

Okay so the second part of the thing is that I am pointing the pointer letter to course 0. So I am pointing it to, remember course 0 is numerical methods, and I am going to point letter into course 0, and then I am going to subtract. I am going to run through a loop, and I am going to increment every time the letter pointer by one step so it will point to the next element, etcetera. So remember, the first time when I said letter, and set it to course 0, course 0 is my string numerical methods, so it will be basically pointing to  $n$ , that is the 1st element, and then I would say letter plus plus, that means, it will point to  $u$ , and I am going to subtract that pointer, okay this pointer course 0 from this pointer, and it will print out the value we get.

(Refer Slide Time: 47:03)



```
bash-2.05b5 gcc sampl3.c
bash-2.05b5 ./a.out
Numerical Methods and Programming
3 0 12
0 N
1 u
2 m
3 e
4 r
bash-2.05b5
```

We get an integer there again, and the value of that pointer, or the value to which it is pointing to okay. That is what we are going to run now. Okay so I comment on that, and I am going to run this. So that is what I am going to run. We are getting okay, so first, its value is 0 and it is pointing to  $n$ , and then I am incrementing the pointer. Remember, I am incrementing the pointer every time. So it keeps pointing to the different letters of that string right okay and when I subtract from that I get the difference between the string and the initial point right. So remember, this is pointing to the first element of the array.

It is a pointer to the first element of that string right. Course 0 is a pointer to the first element of that string. When this was pointing to the same string you get 0, and then as you increment this you keep getting to which element you are pointing to. So that is one of the examples of pointer arithmetic. We now look at pointers to pointers. In the previous section, we saw pointers, which was, pointing to variable memory, and now we are looking at pointer to pointers. That is defined as this one, that is, character, double asterisks, marker.

So marker is now a pointer to a pointer. That means, if I define a pointer like this, for example, this is an array character string here called course 0, and course 1, and course 0 is numerical, and course 1 has got the string methods. So now, we know that course 0 is a pointer also. This is an array. Character string is a pointer also. It is an array, it's a string, character string, it is also a pointer.

So I can use this marker, this pointer, to point to this pointer to read out each of these characters in the string. So that is what we are going to show here okay. So, for example, I make this assignment here, that marker is equal to ampersand course 0. So now, marker is pointing to the base address of course 0. Okay that means, we are pointing to n in course 0, and then if you say so that is to the numerical that is pointing to the base address n, and then we say marker plus plus, now remember marker is now a pointer to this pointer.

So it is pointing to this particular pointer which is course 0. Now when I say marker plus plus, it will point to the next element of that character array. So it is a character array here that is course 0, and then course 1, so if I say marker plus plus, it will then point to the next element of the character array. That will be two methods okay. So that is going from one string to another string but we can also use the same to point to the different elements, and for that we would use this way.

So now, marker plus plus, remember marker plus plus taken this pointer to the next element, which is course 1 okay. And then I will say this asterisk in brackets, asterisk marker plus plus okay, and then it will start moving along the string. So when it is marker plus plus here, it is pointing to the base address of this, that is, it is pointing to m. And when you say asterisk marker plus plus, it will point to the next element, and the next element, etcetera. So actually, it will now point to e, and then if you say marker plus plus again it will point to t, etcetera.

We will see that in a program. Here is a program which shows that. So we have the understand the assignment statements, sorry the compiler preprocessor commands and then here I have declared a character, a pointer called letter. So this pointer is of the type character, and then you can see that it has one asterisk sign here. So it is a pointer. Then I have a string here, character string, here. Character string has 3 elements in it: course 0; course 1; course 2. I can put it, so there are 3 elements possible, course 0, course 1, and course 2. So that is another pointer string. And then I have a pointer to a pointer, that is a double asterisk marker. So then I put in this here, course 0 as the whole string called numerical methods, and course 1 as and programing. So these are two strings, course 0 and course 1 okay. Then I would say that marker, this marker, that is, this pointer to a pointer to point to course 1.

(Refer Slide Time: 50:35)

```
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"
```

(Refer Slide Time: 50:41)

```
#include <math.h>
#include <stdio.h>

main()
{
    float x,y;
    long int i,j;
    char *letter,*course[3],**marker;

    course[0]="Numerical Methods";
    course[1]="and Programming";

    marker=&course[1];
    printf("%s %s \n",course[0],*marker);

    marker--;
    printf("%s %s\n",course[0],*marker);

    marker++;
    for(i=0;i<13;i++) {
        c 24L, 500C
```

That means, it will point to and programing, and then remember this is the way to print out the character string okay. So now, if we say percentage s, then I am printing out course 0, and now star marker. Let us see what this prints out. It prints out here. I will show you what this prints out. So if I run this program it prints out numerical methods and programing. So the first printout here is course 0. So the course 0 is numerical methods, and then we have the star marker. Marker is pointing to course 1, so that is, and programing. So I can print the string out by just writing here course 0, and then saying course 1, or I could use course 0 and say star marker. So marker, you remember, simply marker will not give you that value, there is a star marker, that is actually pointing to course 1.

So in that print statement you will get this entire statement here saying numerical methods and programing. That is one way of printing that out. Now, if I say marker

minus minus, that means it will go back. Now it will point to course 0. It was pointing to course 1 here okay, then it points to course 0, that is numerical methods okay, and then if I say course 0 here, and then I say star marker, that is what this part is. That will print numerical methods and numerical methods twice because now the star marker is pointing to numerical methods, because we set marker minus minus. So that is the way to go forward and backward using a pointer.

So here again, now if I say marker plus plus, it will again take it back to course 1 and that is to point to programing right. So that is the thing, and now once we have this one now, as I said, this will take in to marker plus plus, that is means, it is pointing to now and programing, and then to printout. Now we try to printout each of these characters inside this string, inside this string. That string has course 1 as and programing. So that is what we have, string, and then the gap there, a blank. Okay so we try to print out all of that. So that is the way to print out that.

As we said earlier, star marker plus plus will take us one step forward, and we say star of that star marker plus plus will print out that variable. Now, if you want to print one character you use percentage c okay, and if you want to print out a string then you use percentage s, you remember that. Now I am going through this in a loop starting from i equal to 0 to i equal to 13 in steps of 1, that is, starting from i equal to 0, and then it goes to i less than 13, that means i equal to 12, in steps of 1 and each time it will print the i value and the value which is inside the string, and that is what we would see here. That is what it is printing, i value and the string.

So the 3 rd element, the 4th element, actually, is a blank. And then it is printing out this each of the elements here. So that is up to 12 we have printed okay that will give you all the way up to m, m i. So, before we stop just we read this once again. We were looking at here pointer to pointers. So that is, we can also use this to read out array elements. When we look at arrays, we will use this again. So here we are using now character string which is similar to arrays. I am trying to read out each of the elements of that string by using this using a pointer which points to that pointer.

Then I increment the pointer by using the plus plus that takes it from one array element to another array element. While I want to go through the array, okay I will use that using star marker plus plus. Then if I put another star on that I can print the value which it is pointing to, and that is what we see in this thing here okay. We will stop today is lecture at this point.