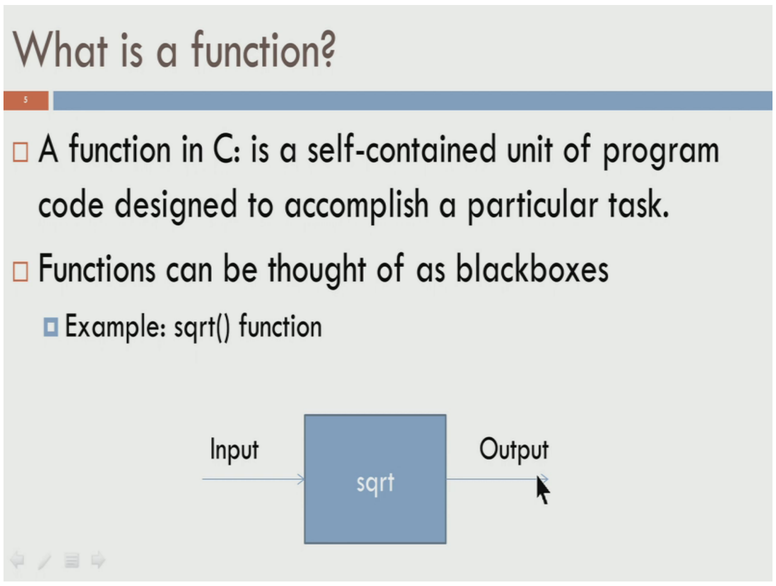
Programming, Data Structures and Algorithms

Prof. Shankar Balachandran
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 10A
Lecture - 20
What is a function? Why use functions
Example: power (base, n)

This lecture on functions, this a very important aspect of C that one has to learn and it also makes a programming much more easier and fun.

(Refer Slide Time: 00:25)



What is a function?

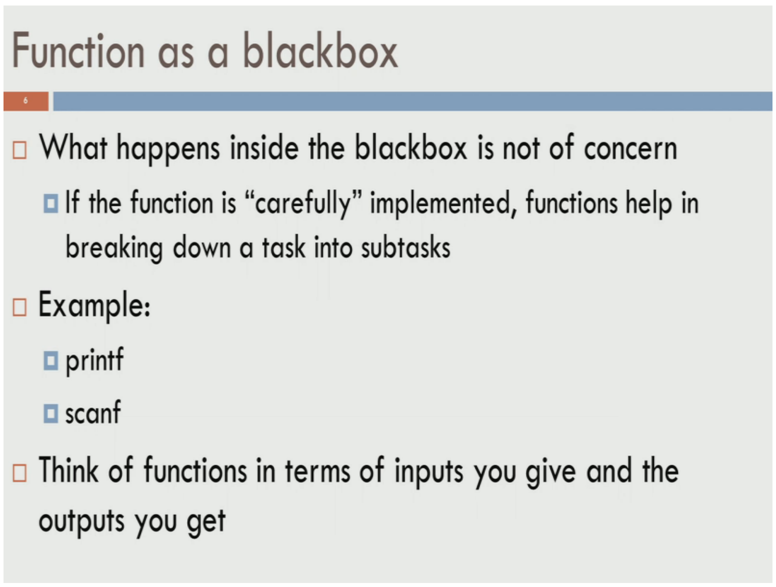
- A function in C: is a self-contained unit of program code designed to accomplish a particular task.
- Functions can be thought of as blackboxes
  - Example: sqrt() function

So, what is a function? A function is essentially a self contained unit of program. It is a piece of code that is designed in such a way that you can accomplish a particular task. So, it is a very specific task that you want to do and you wrap that up in, what is called a function. And one way to look at function is, you can treat them as black boxes. So, for example let us say I have a function called sqrt which supposedly does square root of a variable, you pass a variable to it and it supposed to give you a square root.

So, this kind of thing happens in the mathematical world often. So, I want to find out f of x and so on. So, let us say I as a programmer have very carefully written a program to do square root. And let us say, you do not know how to do it. So, what I do is, I write the program for square root and I put it in what is called a function, I give that as a black box to you and you have to give it inputs I will compute square root for you and I will give you outputs.

So, the program or the sub program that I wrote will compute the square root for the inputs and you will get output. So, this is the notion of a function, so you may not want to know the internal details of square root. For all practical purposes, you can think of this square root as a black box that is given to you. And as long as it is done correctly, you can give inputs, you can expect it to give outputs and you can use that in your calculations. So, this notion of putting things in a black box is nice, because what happens inside the function can be hidden from what happens outside it.

(Refer Slide Time: 02:06)



Function as a blackbox

6

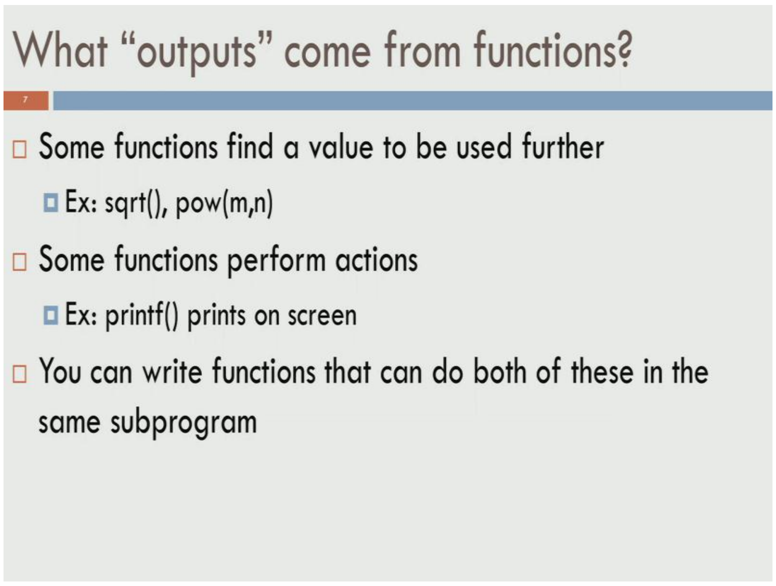
- What happens inside the blackbox is not of concern
  - If the function is "carefully" implemented, functions help in breaking down a task into subtasks
- Example:
  - printf
  - scanf
- Think of functions in terms of inputs you give and the outputs you get

So, what happens inside the black box is not of concern of course, you have to be careful. So, if I say that I given you a program to write square root and let us say that sub program is incorrect, then you are in soup. So, you have to be careful, if the functions are written carefully and implemented correctly, then it let's you take a big task and divide that into smaller sub tasks. And we have seen several examples of functions and there are three things that I can immediately point you to main for instance, it says for function.

Even though, we will talk about main in a little more detail, later. The two functions that we have been using repeatedly are printf and scanf, printf was used to print things on the screen and scanf was used to scan things of a keyboard. So, these two are functions, so clearly you did not write printf and scanf. These were written by someone else and you are using it as a black box. You did not worry about, how things go to the screen nor where you worry about how the keyboard, the key press that is done is taken to the internal variables.

So, this kind of hiding the functionality of what a function does is very useful. Because, you do not have to worry about the inner details of functions, as long as the interface is clean. So, you can think of functions as giving inputs and getting outputs from there.

(Refer Slide Time: 03:34)



What "outputs" come from functions?

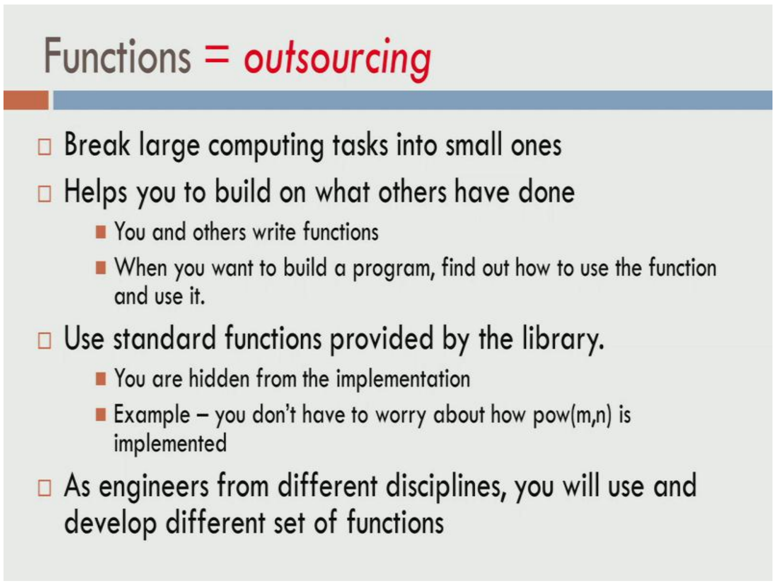
- Some functions find a value to be used further
  - Ex: sqrt(), pow(m,n)
- Some functions perform actions
  - Ex: printf() prints on screen
- You can write functions that can do both of these in the same subprogram

So, let us see what outputs to the function means. So, not all functions actually give outputs, so there are functions which compute something and or expected to return a value. For example, if I am looking for square root, so I want square root of x, x is the value that square root will take and in return, I should expect a value. Because, it has to do some work and return a value. Similarly, there is a mathematical function called pow which stands for power.

Given two floating point values m comma n, it can calculate m raise to the power of n. So, clearly m raise to the power of n is a very useful thing in various aspects and I expect a value more importantly. So, if I give m comma n it can do the action, but I want to use that. Let us say, I want to do a squared plus b squared, I can do power of a comma 2, but that a squared should be returned as a value. So, there are functions which return values.

There are also functions which do not return any values, explicitly or at least not return any useful values, but perform actions. For example, printf is a function, does something to do on the screen. It does not return a value to the program, where printf is called. So, at least it does not return a useful value for all practical purposes. So, you can write functions which does a sequence of actions or you can write functions which does a sequence of actions, compute something and also returns a value.

(Refer Slide Time: 05:07)



Functions = *outsourcing*

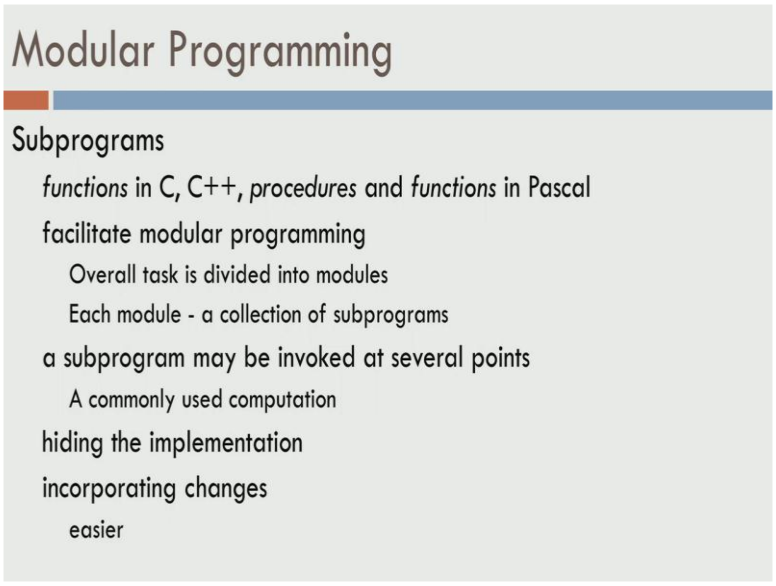
- Break large computing tasks into small ones
- Helps you to build on what others have done
  - You and others write functions
  - When you want to build a program, find out how to use the function and use it.
- Use standard functions provided by the library.
  - You are hidden from the implementation
  - Example – you don't have to worry about how pow(m,n) is implemented
- As engineers from different disciplines, you will use and develop different set of functions

So, the notion of functions is essentially equivalent to outsourcing. Let us say, you are a busy programmer, you are doing something, you do not want to spend time on writing square root function, you outsource it to me, I write the square root function. So, I am good with numerical methods, I go and write the square root function and I give it to you and that way, you can outsource your work. So, you can take a big problem, divide that into smaller problems and compose this smaller sub task together.

This is very useful, when you want to build large programs. You can build on top of what you have done earlier or you can also build on top of what others have done. So, this is a very nice thing about using functions. So, most languages give you, what is called a library a C also has a very rich set of libraries. And very common things, the most common things the most programmers would need are all put in the library.

So, imagine every single user or every single programmer having to know, how to write things on the screen, having to know how to read from the key board and so on. Instead, C as a language gives you printf and scanf. Similarly, there are several mathematical functions including pow and square root and so on, which are all packaged in what is called a library and as engineers, you will actually either develop these libraries for others or you may actually end up using libraries that others have written.

(Refer Slide Time: 06:40)



Modular Programming

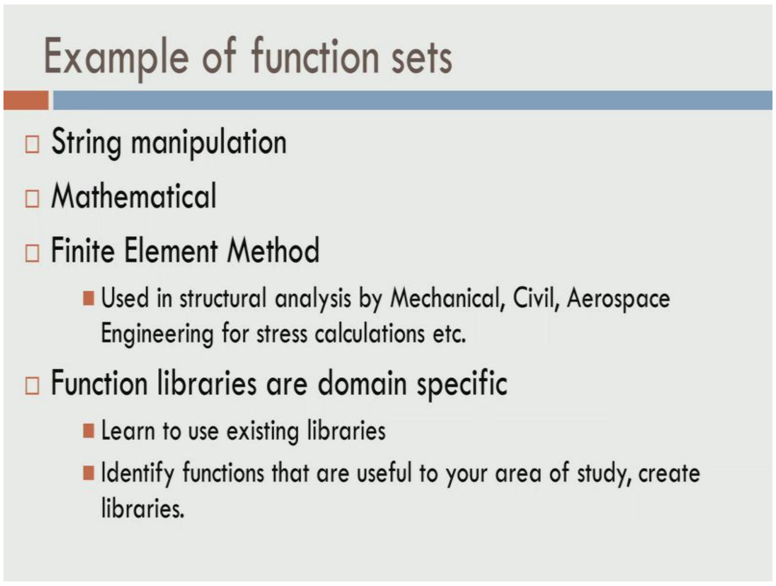
Subprograms

- *functions* in C, C++, *procedures* and *functions* in Pascal
- facilitate modular programming
  - Overall task is divided into modules
  - Each module - a collection of subprograms
- a subprogram may be invoked at several points
  - A commonly used computation
- hiding the implementation
- incorporating changes
  - easier

So, one thing that you get from using functions is what is called modular programming. So, you take a task, divide that into sub task as sub programs and the sub programs can either be things that do some work or things that do some work and return a value. So, if you have been exposed to languages like Pascal, in Pascal they called them procedures and functions. Procedures do not return values, whereas functions return values.

In C and C++, everything is called a function, if a particular function does not return any value, it does not have a special name for it. It facilitates modular programming, you can take this sub program or the function that you have and you can invoke it at several points. So, I will show a simple example later, on how this becomes modular and as I said, it hides the implementation and if you want to make any change to the program, it makes it much more easier.

(Refer Slide Time: 07:40)



Example of function sets

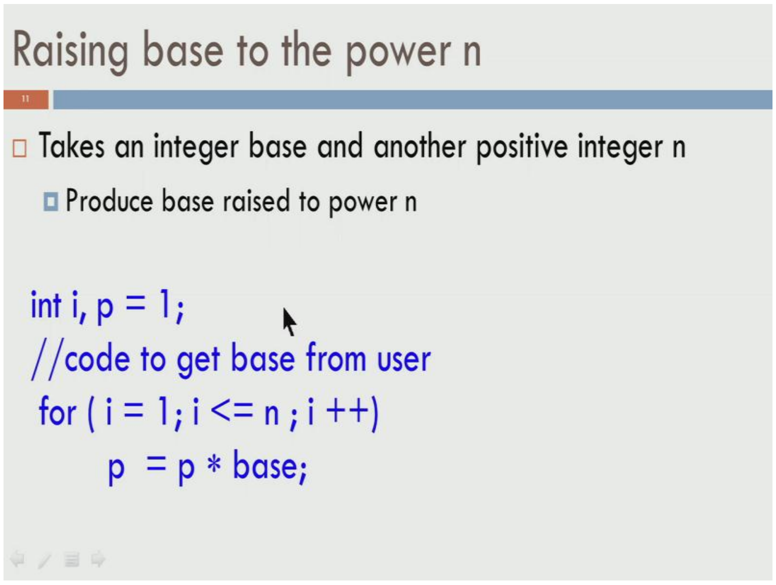
- String manipulation
- Mathematical
- Finite Element Method
  - Used in structural analysis by Mechanical, Civil, Aerospace Engineering for stress calculations etc.
- Function libraries are domain specific
  - Learn to use existing libraries
  - Identify functions that are useful to your area of study, create libraries.

So, what kind of functions are there in C? There are several functions available in C for string manipulation, this is a very common thing. How do you take a sequence of characters forming a string and manipulates strings? So, we will look at strings in a later lecture. There are also several mathematical functions that are available as standard libraries in C. So, if you are coming from other engineering disciplines, so you develop your own things.

For example, Mechanical Engineers, Civil Engineers, Aerospace Engineers and so on, may use finite element methods, very often and there are libraries that are available. Not a standard libraries, but there are vendors who provide these libraries. Many of these libraries are usually domain specific. So, what I mean by that is, so somebody use a specialist in an area requires a specific kind of computation. This may not be useful to people outside the domain.

For example, finite element methods are useful to certain class of engineers, whereas somebody in Electrical Engineering or Computer Science may not really need finite element methods or libraries doing that in any of the regular routine job. So, as a programmer what you will do is, you will learn to use existing libraries and you may also identify scope for building new libraries and you may even make money by building these libraries and selling them later.

(Refer Slide Time: 08:50)



Raising base to the power n

- Takes an integer base and another positive integer n
  - Produce base raised to power n

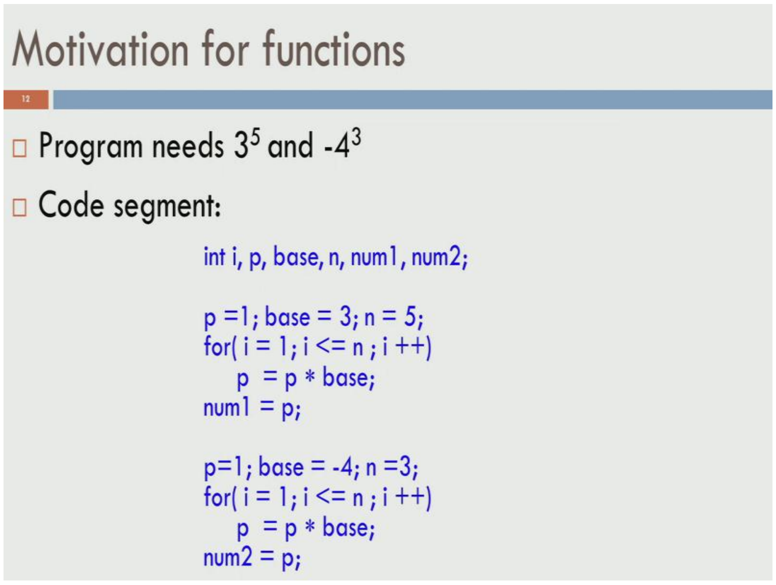
```
int i, p = 1;
//code to get base from user
 for ( i = 1; i <= n ; i ++)
       p  = p * base;
```

So, let me show you a small piece of code in which I am doing this basic mathematical function, raising x power y. So, let us say we have a number called base and we have a number n and we want to raise base to the power n. So, we will assume that n is an integer greater than equal to 0 and base is any integer, it can be negative, 0 or positive. And there is a small code in this slide which achieves what we want to do, so raising base power n.

So, if you see that there are two integers i and p and p is actually initialized to 1. This is important here and you have a for loop which runs 1 less than or equal to i plus plus. So, this loop runs n times and we take p, multiply it with base n times. So, the first iteration you would get one times base which is base power 1. The second time, you will get base into base which is base squared and third time, p will have base squared into base, you will get base cubed and so on and you will get base power n at the end of the loop.

So, clearly if n is negative then you have a small issue. So if n is negative, then any integer raise to a negative power will be a fractional value and this is not a program which takes care of that. However, if n is 0 then this loop will never execute and p is 1 is actually correct. So, raising in base to the power 0 is 1, it is correct and for anything which is greater than 0, it will run as many times as n is and at the end p will be base power n. So, this is a simple iterative structure, in which we have calculated base power n.

(Refer Slide Time: 10:51)


Motivation for functions

12

- Program needs 3^5 and -4^3
- Code segment:

```
int i, p, base, n, num1, num2;

p =1; base = 3; n = 5;
for( i = 1; i <= n ; i ++)
    p  = p * base;
num1 = p;

p=1; base = -4; n =3;
for( i = 1; i <= n ; i ++)
    p  = p * base;
num2 = p;
```

Let us look at the motivation for functions. Let us say, a programmer needs 3 power 5 and minus 4 power 3 in the same program. So, as a programmer one could write this, so I have i and p and base and n. These are supposed to be four variables which take care of base, n, i and p. So, i is for running the loop, p is for collecting the product and base and n are the two numbers. You are doing base power n, so the base and n are the two integers and the result is supposed to be gathered in num1 and num2.

Num1 is supposed to be 3 power 5 and num2 is supposed to be minus 4 power 3. Let us say, this is the setup that you have, you have to write a program of this kind. So, you setup p equals 1, setup base equals 3 and n equals 5 and you write a loop which takes care of raising 3 to the power of 5 and you make num1 equals p. So, num1 at this point has 3 power 5. So, then you have to again change p to 1, setup the base to minus 4 and setup n equals 3.

In this case, you are ready to do minus 4 power 3 and again you have repeated the same loop that you have earlier. So, you look at this, this loop here and this loop here are exactly the same, the result is accumulated in p and you copy the value of p to num2. So, this is the correct program, as in num1 and num2 would get 3 power 5 and minus 4 power 3, but let us look at a few things. First of all, you have duplicated effort.

So, there is a for loop that is written here, there is also a for loop that is written here. Let us say for some reason I made a small mistake here, this mistake could also come here.

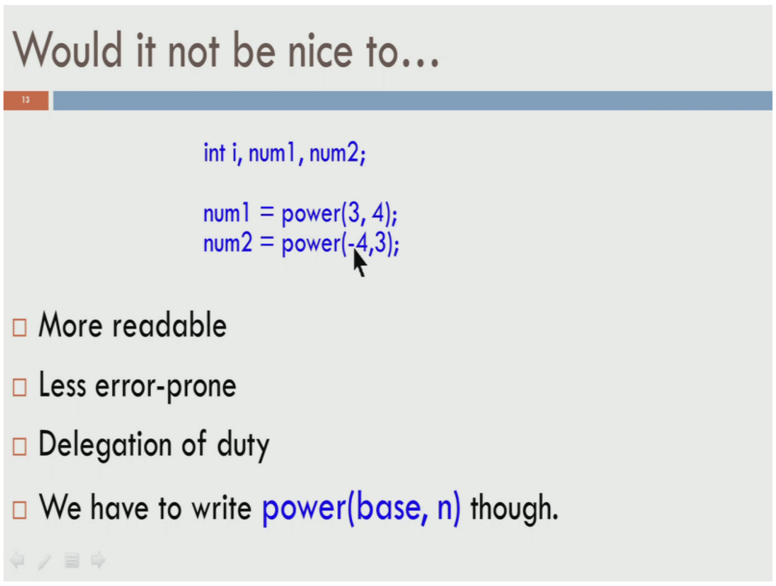
Even, if we have written the program correctly, it is just that the same task that I am doing, I see it in my program over and over and over, that is one thing. Two, this is something very suttle and you may not catch it if you are not careful.

So, even though this code, this for loop is repeated. Let us say, I even let you do that and let us say, it is not a problem right now. Let us look at base and n, anyway these are things that are supposed to change. So, I have setup base and I have setup n, so this is, but let us look at p. So, I have p equals 1 and then we do the set of computations, it gives us 3 power 5. If I forget to do this p equals 1 here, what can happen is I have 3 power 5 that goes to p and then let us say, I forget to do p equals 1.

I have base equals minus 4 and 3, n equals 3. If I forget to initialize p to 1, then p will start with 3 power 5 and to that you are going to multiply minus 4, 3 times. So, you are not really calculating minus 4 power 3 anymore and the result would have been for num2 3 power 5 into minus 4 power 3, this is a disaster. So, every time you want to do this, you also have to remember that p has to be initialized to 1.

Of course, you would have done, change the base and number, because you are computing for something new, but you may forget initializing p to 1 and this can be a recipe for disaster. So, you have to be careful and this is not a very good way of doing it, it is definitely sounds clumsy and we have to change this.

(Refer Slide Time: 14:22)



Would it not be nice to…

```
int i, num1, num2;

num1 = power(3, 4);
num2 = power(-4,3);
```

- More readable
- Less error-prone
- Delegation of duty
- We have to write power(base, n) though.

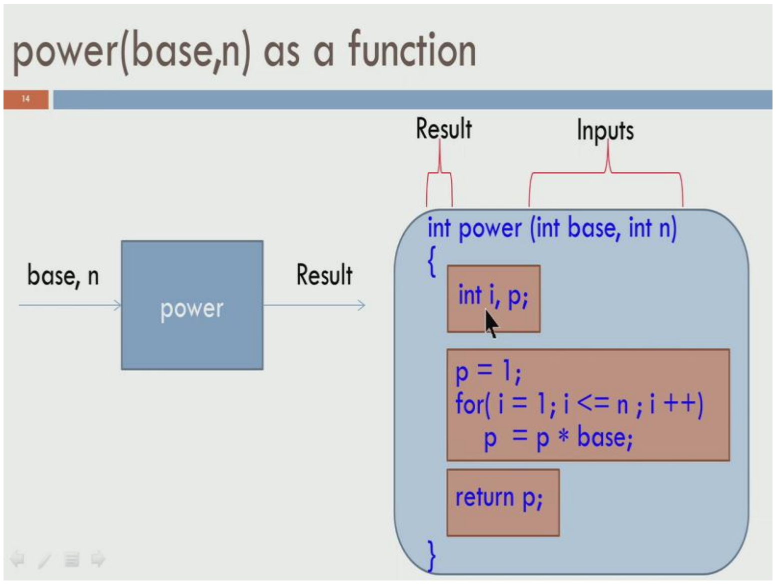
It would be really nice to do something like this, would it not? So, int i, num1, num2, 3 integers, num1 is power of 3 comma 4, num2 is power of minus 4 comma 3. So, this is nice for multiple reasons. One, is definitely more readable, instead of, that screen full of code that you saw earlier, you now have three lines of code, int i, num1, num2, it is for declarations and num1 and num2 gets 3 power 4 and minus 4 power 3.

This looks almost like writing mathematical equations on a piece of paper. So, you have taken 3 power 4, put it to num1, minus 4 power 3, put it to num2 and so on. It is definitely more readable, it is less error prone. Because, if somebody you have essentially outsource the work of doing this power, it is computing power to someone else or some other functionality, you have made a sub task which is supposed to be computing power of base comma n and when you do that, you are not going to run into this problem.

So, every time the function is going to be called, will assume that all the initializations are properly taken care of, the computation is done carefully and you actually get work done and you get the return value. So, if power is a function if it takes these two things and if it returns a value, then you have essentially delegated the duty of computation to either someone else or to some other region of the code. Definitely, this looks much less clumsy.

So, the only thing is somebody still has to write, this power of base comma n. So, you are outsourcing it, but at the end somebody has to write it.

(Refer Slide Time: 16:10)



power(base,n) as a function

```
int power (int base, int n)
{
    int i, p;

    p = 1;
    for( i = 1; i <= n ; i ++)
        p  = p * base;

    return p;
}
```

So, let us see how this power of base comma n can be return up as a function. I said, when you use it, you can think of it is a black box. So, power is a black box, you get base and n as inputs. If you give these as inputs, you are supposed to get some result back, which is supposed to be base power n. So, let us start with this black box, so what I am going to do is, I define body. So, I have this body with this left and right set of flower braces and I have a name for a function called power. This is the name that I am going to use for power.

It takes two integers as parameters, int base and int n. This takes care of the inputs and I am supposed to be getting a result back and whatever is return from a function, is what you have here. So, the way to look at this line is the function is called power, it takes two inputs base and n and it returns an integer. So, the return does not have a name, it just has this type. You take two integers as inputs, one integer is called base, another integer is called n and we get an integer as a result.

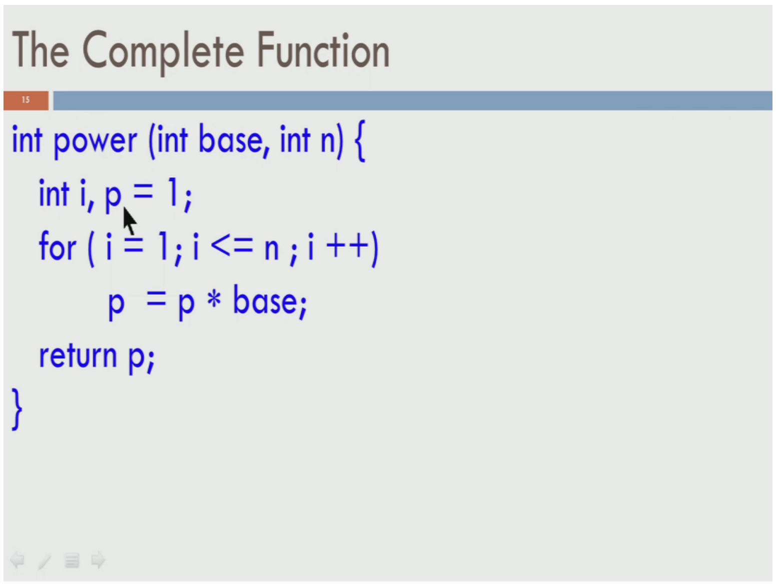
So, that is what you have so far, so this is what the black box setup is. So, clearly you have to write the program for it, we did this earlier, so you already had it. So, p equals 1, for i equals 1, i less than or equal to n, i plus plus p equals p times base. So, this actually does the computation. So, we already have this program written up and let us say, I just did cut and paste of this into this block of code. So, I have this, having this is not sufficient.

So, if I look at this as a sequence of steps, I have variable p variable i, but these variables are never declared. So, there is no declaration for p and i and I mentioned earlier that every variable has to be declared, so that you can get a location in memory. So, p and i are not declared yet, so let us do that first. So, int i comma p, you have that, so this is not the end of it, you have declared int i and p at the end of this for loop, p will actually have the final result.

But, if you look at this function, there is a variable i, there is a variable p, there are also these two inputs that you took base and n. So, I want a result back, but which one of these variables is the result. No way, I compile and I can figure out, which one of these things should be returned as a result. So, for that we use this keyword called return, so this return, return is a keyword. So, it is a reserved word in the C language and what it does is, after various variables that and various calculations that you have done, the final result is stored in p, please take that and return it to, whoever called it.

So, if we have base comma n as an inputs, there are lots of local variables inside power. There are variables like p, i and so on and of these, you are expecting p to be returned and return p takes care of that.

(Refer Slide Time: 19:44)



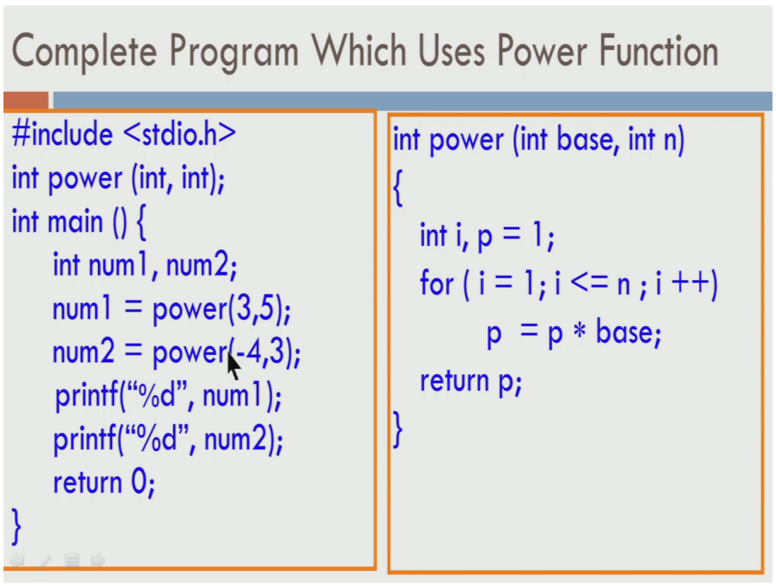
The Complete Function

```
int power (int base, int n) {
  int i, p = 1;
  for ( i = 1; i <= n ; i ++)
        p  = p * base;
  return p;
}
```

So, essentially if you look at the whole program, the whole function this is how it is, int power of int base comma int n. So, from by looking at this I know that it is expecting two inputs and both are supposed to be integers and it is going to give one output, which is also an integer. And which integer does it return? It is going to return p, after doing this set of calculations. So, now let us see how this is nice, so since p equals 1 is an initialization done here.

The key thing is every time this function is going to be called, this variable p is going to get initialized to 1. So, therefore you do not have to remember to initialize p every time, you have given the task to somebody else. So, you have delegated the task and that task or the subtask here called power, is supposed to take care of whatever is required to compute power of base comma n. So, to put this whole thing in one package, let us see a complete program which uses the power function.

(Refer Slide Time: 19:56)



Complete Program Which Uses Power Function

```
#include <stdio.h>
int power (int, int);
int main () {
    int num1, num2;
    num1 = power(3,5);
    num2 = power(-4,3);
    printf("%d", num1);
    printf("%d", num2);
    return 0;
}
```

```
int power (int base, int n)
{
   int i, p = 1;
   for ( i = 1; i <= n ; i ++)
        p  = p * base;
   return p;
}
```

So, what you see on the right side is the function, you have int power, int base and int n and we have written this function and let us see, how this can be used. So, we have two integers, num1 and num2 and I said, would it not be nice to have num1 is power of 3 comma 5 and num2 is power of minus 4 comma 3, we have exactly that.

So, this is an actual program, we have these set of lines followed by these set of lines, let us say. So, even though it is all supposed to be in one place, one after the other I am showing it right and left. So, let us say I have these set of lines followed by this set of lines. So, let us see, so we have num1 is power of 3 comma 5, num2 is power of minus 4 comma 3. So, at this point power of 3 comma 5, so that would be 243, I would expect num1 to have 243 and power of minus 4 comma 3 is minus 64, negative 64. So, that supposed to be num2.

So, once I have computed it, I can actually print it. So, in this case it is only printing, you may want to use it in other calculations, later also and we have completely delegated the work of power. So, this whole thing is very clean and readable, so in this context I want to give some names. So, at this point we say that the power function is being called at this point and this point and power is a function here, so this is called the caller.

So, the main function here is called the caller. The caller calls power function twice and power is the callee. So, you have a caller which is main and you have a callee, which is power and the caller can call callee, multiple times. So, that is what I said earlier, so you may have some task which is repeatedly done. So, as long as you have written up the callee function and it is cleaned up, it is correct and you have verified it and so on, you can call it as many times as you want in any caller. So, we will see more details of functions in the subsequent modules.