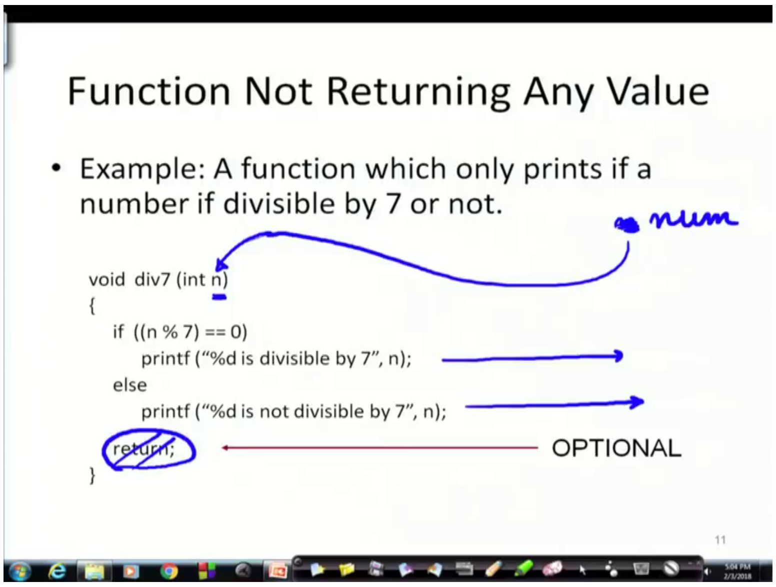
Problem Solving through Programming In C
Prof. Anupam Basu
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture – 37
Functions (Contd.)

So, in the earlier lecture, we have seen how a function can be invoked, and it returns the values. As we had mentioned that it is not the case that always a function will return a value here is an example.

(Refer Slide Time: 00:35)



Function Not Returning Any Value

- Example: A function which only prints if a number if divisible by 7 or not.

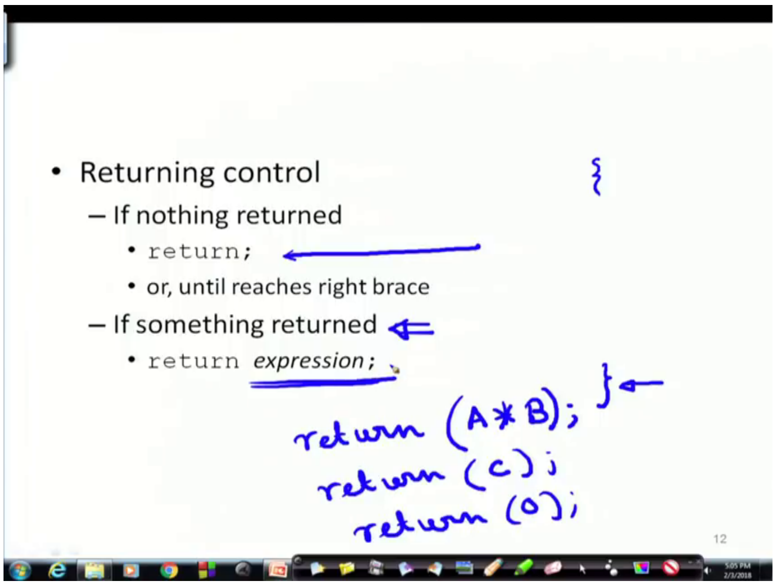
```
void div7 (int n)
{
    if ((n % 7) == 0)
        printf ("%d is divisible by 7", n);
    else
        printf ("%d is not divisible by 7", n);
  ~~return;~~          <------------------  OPTIONAL
}
```

You look at this function div 7 which says; that means, the purpose of this function is to find out whether a particular number that is passed on to it as a parameter, and parameter whether that is divisible by 7 or not. So, the program the code is very simple what should we do? If n is divisible by 7; that means, if n modulus 7 is equal to 0, then we say printf n is divisible by 7 otherwise we print n is not divisible by 7.

Now, in this case we are just printing from here straight the print out is coming printing is coming out from here. The main function is only passing on the value reading scanning reading some value of n and passing on that value of n here or may be here it is n is the argument. So, might be the main program is reading a value num, and passing that value num here. Now this num is also of type integer and the rest of the things are being done by the function. It is testing whether it is divisible by 7, if it is divisible by 7, it is printing like that otherwise its printing the other message. Now in this case putting this return is optional, because even if I did not put the put the return when I would have met these parentheses that is n bracket it would automatically return to the calling point.

However there is no harm if I put the return.

(Refer Slide Time: 02:44)



- Returning control
  - If nothing returned
    - `return;`
    - or, until reaches right brace
  - If something returned
    - `return` *expression*;

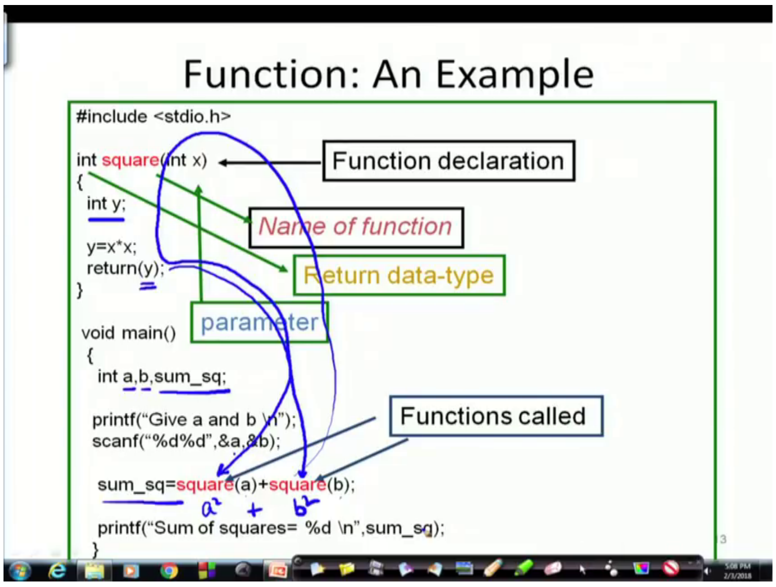
return (A*B); }

return (C);

return (0);

So, returning control; we have seen that how the thing is invoked by parameter passing. Now returning control if nothing is returned then you can simply write return semicolon or we can skip that and until it comes to the right brace, that is last right brace that is automatically taken as the return. But if something is to be returned if something is to be returned then we must put the return statement with say may be return A times B some expression or it could be return C or it could be something like return 0 or return 1 whatever we have to do something, some expression and expression automatically in I mean you know we will also capture the constants so.

(Refer Slide Time: 03:57)



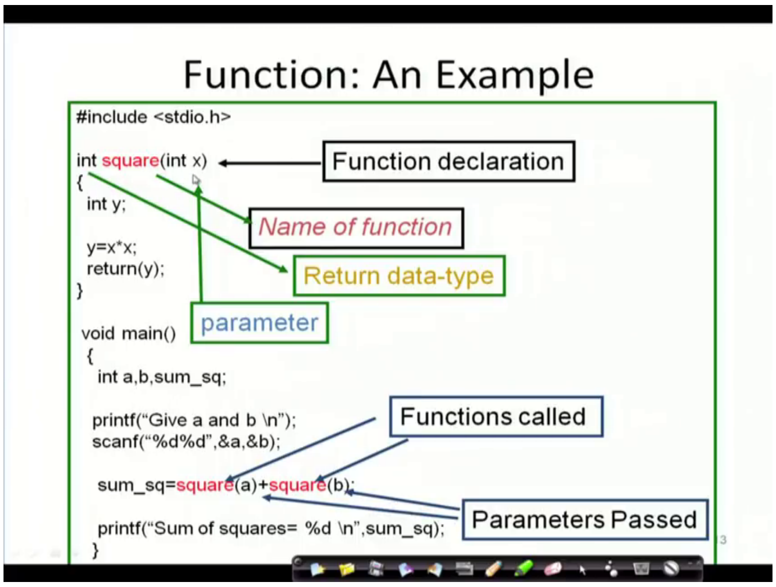
Function: An Example

```
#include <stdio.h>

int square(int x)
{
  int y;

  y=x*x;
  return(y);
}

void main()
 {
  int a,b,sum_sq;

  printf("Give a and b \n");
  scanf("%d%d",&a,&b);

  sum_sq=square(a)+square(b);

  printf("Sum of squares= %d \n",sum_sq);
 }
```

Now, let us look at this example here you see the layout is also important here we are first writing the function this is the function the function declaration is coming first square of x square of an integer x and square of an integer x will also be an integer therefore, the type of the function is int as you can see here. So, that is this whole thing is the function declaration, then starts the body of the function. So, here the function declaration consists of the name of the function, and the parameter that is x of type integer and this int, here is a return data type what the type of the data type that will be returned. So, these 3 together make the function declaration.

Next we are coming to the body of the function int y what is that that is a temporary variable temporary variable why I am calling it temporary? Because it lives as long as this function is active, as soon as the function is function ends the role the definition of this y is also lost. So, here you see we will come to that later that here you see that this is an internal variable, here I am computing the square y assigned x times x, and I am returning y. I am returning y and after returned that y vanishes y where is y returning to? Wherever they square has been called now here you see here is a sum of square the what is main doing now let us come to main. Main has got some variables a b and sum of square is another variable. So, printf give a and b I am reading a and b. Now I am calling this function twice first with the parameter a next with the parameter b and a and b how I mean in sequence goes to this argument x and the square of a is computed. So, return y will return first here a square. So, we get a square here then this is called and y is returned here. So, we get b square and then these 2 are added and we get sum of square alright and then we are printing the sum of square. So, you need to you can also try to see what is a flow of data in such cases alright let us move ahead.

(Refer Slide Time: 07:28)

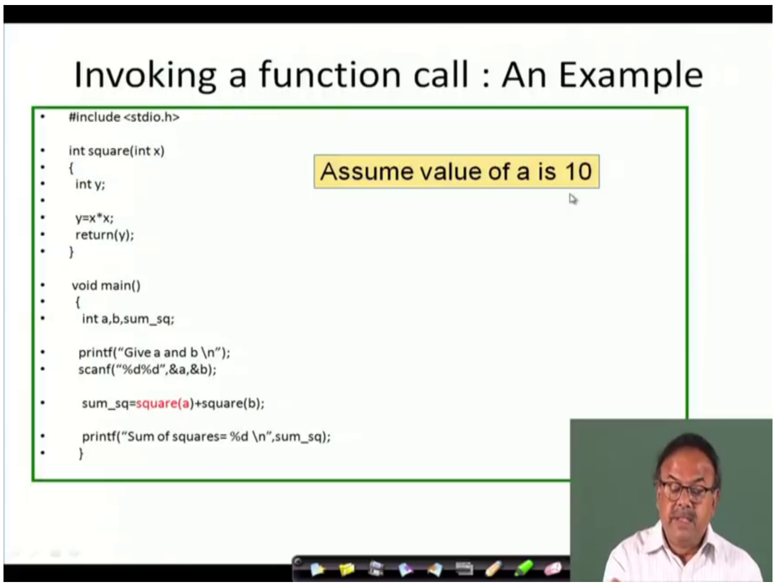


Function: An Example

```
#include <stdio.h>

int square(int x)
{
  int y;

  y=x*x;
  return(y);
}

void main()
 {
  int a,b,sum_sq;

  printf("Give a and b \n");
  scanf("%d%d",&a,&b);

  sum_sq=square(a)+square(b);

  printf("Sum of squares= %d \n",sum_sq);
 }
```

So, these are the parameters passed and here are the, here is the argument you can see that.

(Refer Slide Time: 07:35)



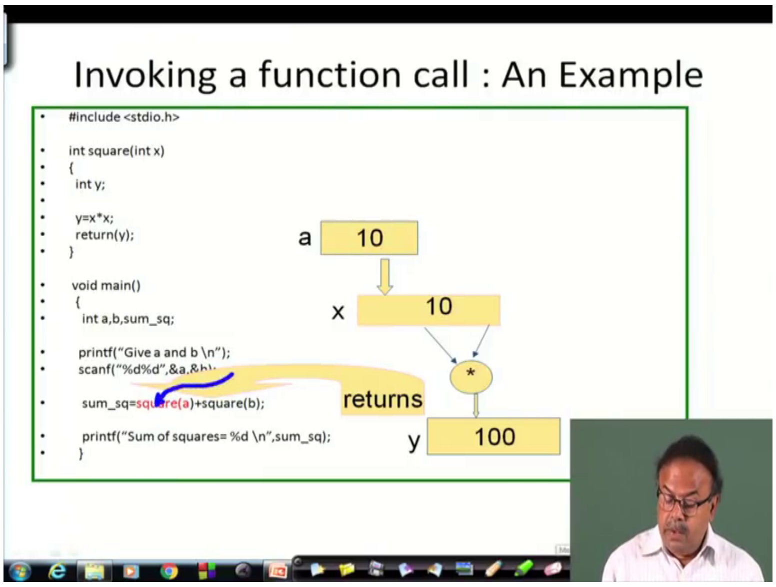
Invoking a function call : An Example

```
#include <stdio.h>

int square(int x)
{
 int y;

 y=x*x;
 return(y);
}

void main()
 {
  int a,b,sum_sq;

  printf("Give a and b \n");
  scanf("%d%d",&a,&b);

  sum_sq=square(a)+square(b);

  printf("Sum of squares= %d \n",sum_sq);
 }
```

Now, invoking a function call here the same thing what is happening when a the thing that I just now explained.

When I am saying now let us see what happens to the variables assume that the value of a that has been read here is 10 alright then square of a means square of 10. So, 10goes to x, right.

(Refer Slide Time: 08:05)

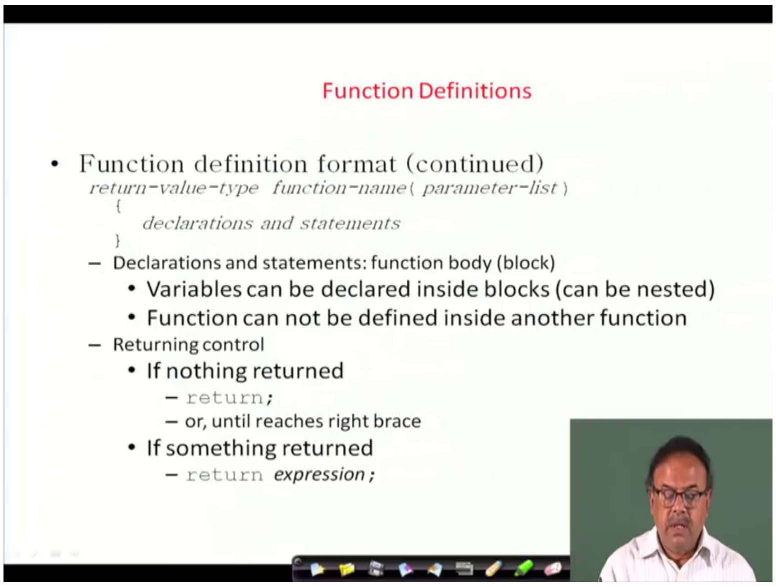


Invoking a function call : An Example

```
#include <stdio.h>

int square(int x)
{
 int y;

 y=x*x;
 return(y);
}

void main()
{
 int a,b,sum_sq;

 printf("Give a and b \n");
 scanf("%d%d",&a,&b);

 sum_sq=square(a)+square(b);

 printf("Sum of squares= %d \n",sum_sq);
}
```

So, a is 10 here that goes to x. So, x becomes 10, now we compute y which is 10 times 10 I am getting y. Y is becoming hundred right now this hundred is coming here actually this arrow is a little wrong here. So, it will be actually coming to this point clear next suppose.

(Refer Slide Time: 08:53)



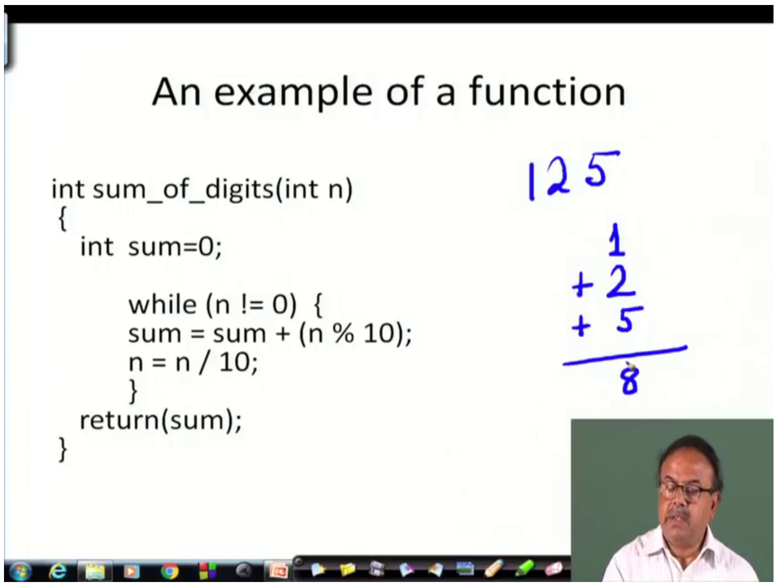
Function Definitions

- Function definition format (continued)
  return-value-type function-name(parameter-list)
  {
  declarations and statements
  }
  - Declarations and statements: function body (block)
    - Variables can be declared inside blocks (can be nested)
    - Function can not be defined inside another function
  - Returning control
    - If nothing returned
      - `return;`
      - or, until reaches right brace
    - If something returned
      - `return` *expression*;

So, in that case, similarly it will be for b, if b was some value that is the how then x suppose b was 7 then x will get 7 and then y will be 49, and 49 will come to that square of b. So, 100 plus 49 will now be added and will be kept as the sum of square. So, in the earlier example you could see that in the earlier example the first the function was return and then the main now let us look at go ahead function definition.

So, we have seen that a function name preceded by return value type and declaration statement and then the function body I am repeating certain things variables can be declare declared inside the blocks the blocks can be nested; that means, there can multiple blocks function cannot be defined inside another function this must be clearly understood a function cannot be defined within another function and returning of control the control will have to be returned as we have seen, if nothing returned then return we have already seen that if something is returned then return that expression.

(Refer Slide Time: 10:24)

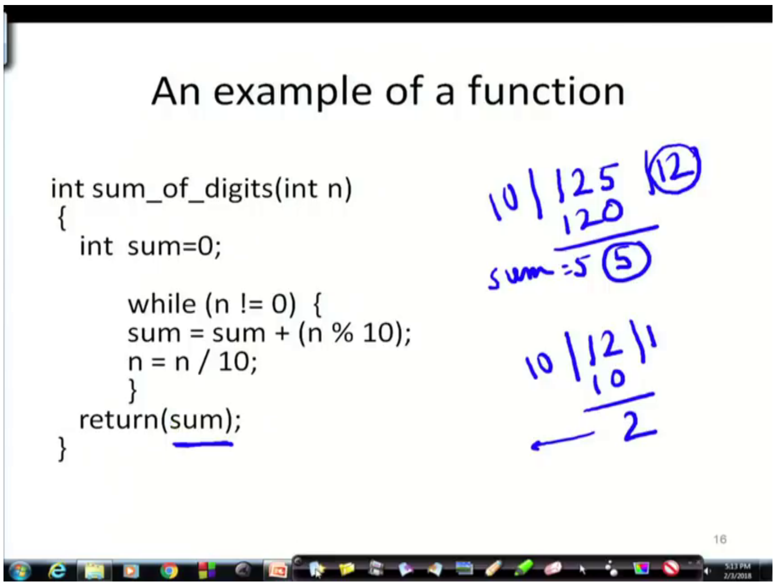


An example of a function

```
int sum_of_digits(int n)
{
  int sum=0;

      while (n != 0) {
      sum = sum + (n % 10);
      n = n / 10;
      }
  return(sum);
}
```

Here again another example of function, the more examples you do the better you will understand. Here the function as the name implies you see its always better to use meaningful names of functions some of digits of n. So, if there be number like 125 then I am trying to extract this digits 1 plus 2 plus 5. So, that will be 8 that is what my program wants to do. So, initially sum is equal to 0 while n is not equal to 0, sum plus remainder.

(Refer Slide Time: 11:17)



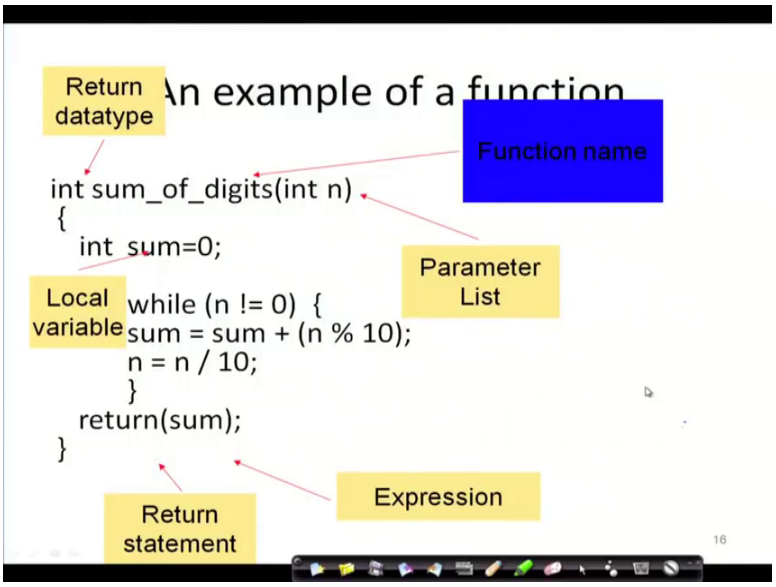
An example of a function

```
int sum_of_digits(int n)
{
  int  sum=0;

      while (n != 0)  {
      sum = sum + (n % 10);
      n = n / 10;
      }
  return(sum);
}
```

So, 125 if I divide by 10, then I will have 12 here and remainder is 5. So, sum plus 5. So, sum becomes 5 clear.

Now, then we find out the device dividend that is 12 and again divide that by 10, we get 2 to be the remainder. So, we take sum to be 5 plus 2 and so and so forth. Ultimately I am getting the sum. So, return sum this another example of a function; here you can see that this n is coming as a parameter all these sum; sum is a an internal variable and will not have life beyond the body of the program.

(Refer Slide Time: 12:21)

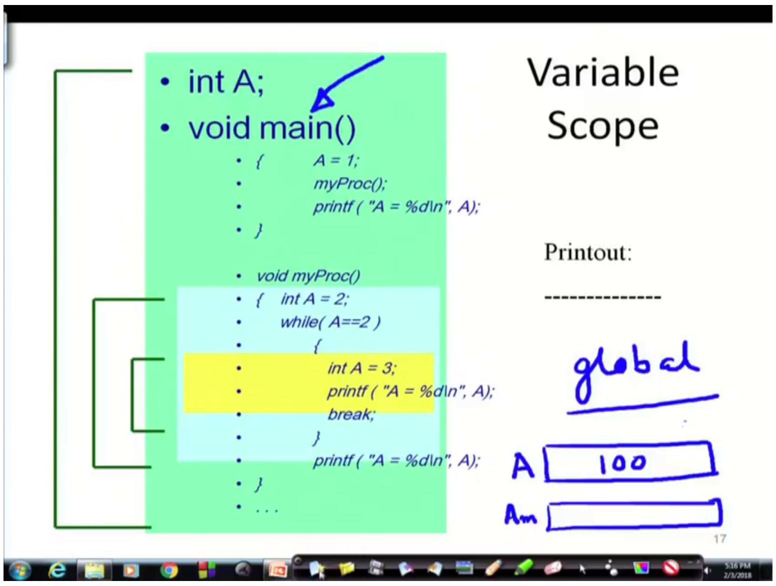


An example of a function

```
int sum_of_digits(int n)
{
  int  sum=0;

      while (n != 0)  {
      sum = sum + (n % 10);
      n = n / 10;
      }
  return(sum);
}
```

So, here you see sum of digits is a function name int is a return data type, parameter list is that local variables is sum. Sum is a local variable and return statement is return sum clear all these we have already discussed, this is merely a division; and here you can see that the return can have an expression here only a variable is an expression.

(Refer Slide Time: 12:54)



Variable Scope

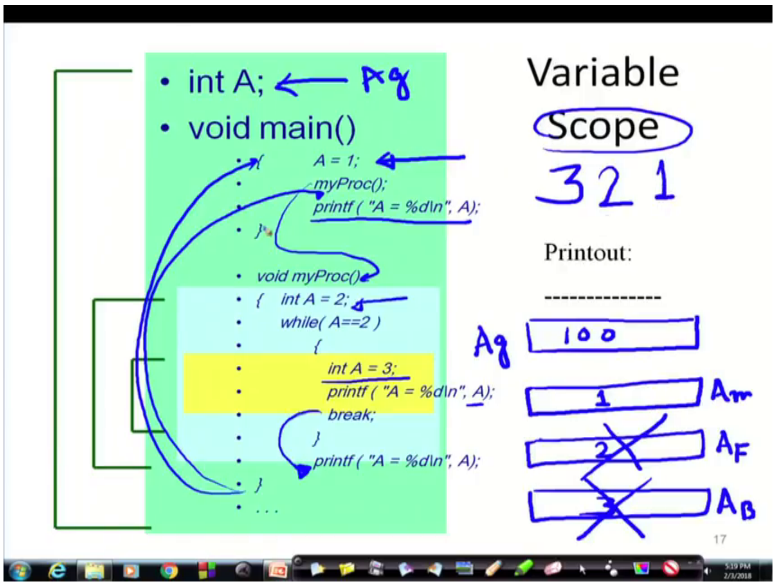
```
• int A;
• void main()
    • {     A = 1;
    •       myProc();
    •       printf ( "A = %d\n", A);
    • }

    • void myProc()
    • {   int A = 2;
    •     while( A==2 )
    •       {
    •         int A = 3;
    •         printf ( "A = %d\n", A);
    •         break;
    •       }
    •       printf ( "A = %d\n", A);
    • }
    • . . .
```

Printout:

Now, we come to a very important concept, which I was mentioning in the passing that the life of an intern variable internal to a function exists as long as the function is live as long as the function is active.

Now, that formally is known is called the scope of the variable or the variable scope. So, let us look here there is an interesting program. Now here you see my entry point is the main, but even before that, I have declared a variable int A; that means, this A is a global variable that means, it is there always. Suppose it is A value 100 then it remains this this is retained. So, let us keep this and let us see what happens. Now I am entering main and I have assigned A to 1 now; that means, now inside this function inside this main I have got mains A which is another A let me write it m is it visible let me do it in a better way.

(Refer Slide Time: 14:38)



Variable Scope

```
• int A;
• void main()
  • {       A = 1;
  •         myProc();
  •         printf ( "A = %d\n", A);
  • }

  • void myProc()
  • {   int A = 2;
  •     while( A==2 )
  •         {
  •             int A = 3;
  •             printf ( "A = %d\n", A);
  •             break;
  •         }
  •         printf ( "A = %d\n", A);
  • }
  • . . .
```

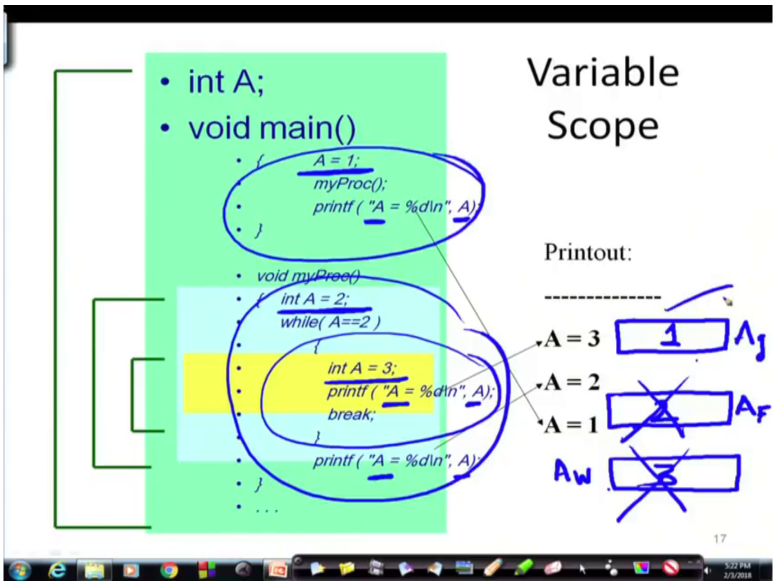
Printout:

So, there is one A, I am calling that A g that is this A, is A g that is the global A and suppose that is 100. Now this is another a which is defined in the main. So, the function this will be live inside main. So, let me call it A m alright. So, now, A becomes 1 here as I come here a 1, then I am calling my proc a function my proc from here it comes to my proc and suppose my proc has got another A here. So, let me call that to be A x just for understanding that it is of my proc or let me call it let me call it of the function right; so, AF. Now that is initialised to 2 now you see how many different a's I have? Multiple a's now inside this block I initialise another A to be 3. So, that is another A that means, this one is not being disturbed A is 2 and while A is 2 I am making another a because here I am declaring you see this is a pure declaration int A. If I had just written A, if I had just written A assigned 3, then this A would be assigned 3, but here I have declared another A int A.

Therefore there is another A coming within this y while block that I am calling AB, and that is becoming 3. Now I am printing A, which here will be printed the inner most the current A that means, 3 will be printed then I break; break means what? I come out of the while loop we have learned break. So, we come out of the while loop and then I print a which is so, as soon as I break out of this this is gone no longer live.

So, I am coming here now which a is in my scope which a is in my scope, my scope is this I am within this function. So, this A. So, that will be printed here. So, 2 or there was some back slash n I am ignoring them that will come one after another and then I come out of this its over. So, I go to this main function go back sorry not here I am sorry it should go back to this point; that means, I will now execute print which a is in my scope now I have come out of this come out of this function. So, the its scope is also gone. So, the scope of this function main is now live. So, what is the value one? So, that will be printed; that is how the things will be printed. So, we will repeat it if necessary, but let us try to see the execution.

(Refer Slide Time: 18:14)



Variable Scope

```
• int A;
• void main()
    • {   A = 1;
          myProc();
    •     printf ( "A = %d\n", A);
    • }

    • void myProc()
    • {  int A = 2;
    •    while( A==2 )
    •    {
    •       int A = 3;
    •       printf ( "A = %d\n", A);
    •       break;
    •    }
    •    printf ( "A = %d\n", A);
    • }
    • . . .
```

Printout:

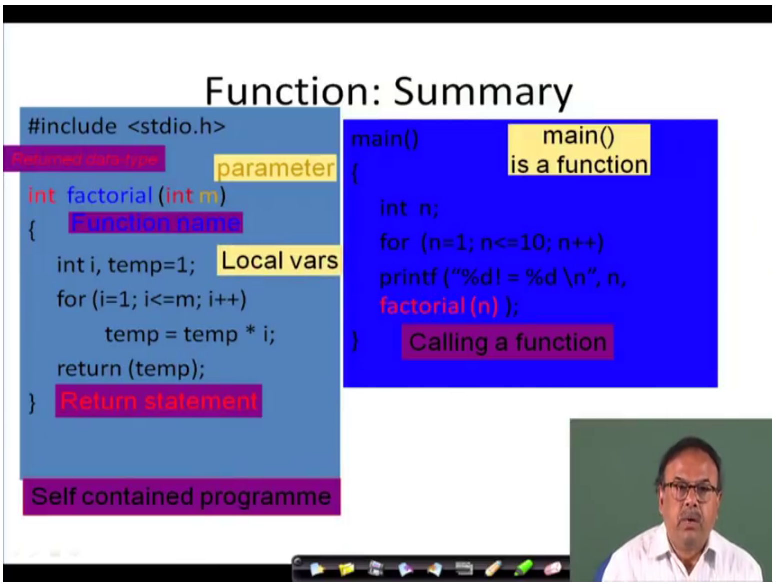
```
--------------
A = 3
A = 2
A = 1
```

Now so, if I first do it then this one will be printed a assigned 3 next that is gone. So, here A will be this will be assigned A assigned 2, then I will go up there and the A that is in the scope of this function that will be assigned and then. So, that will be assigned ok.

You see although I declared a global A internally when I declare some other A, this global A I have I look here a point has to be seen. I declared A, a global A here A g that was declared. Here I have assigned to A, I have not declared another A, I have not written int A I have simply written a assigned 1 that means, the a that was there is already existing globally that has been assigned one.

But when I come here and I am declaring int A internally inside this process, another A is created which is the A of the function and that is assigned by with 2 not this one, they are 2 distinct entities. Now again here I have declared another A. So, since I have declared another A, this is the A of this while loop might be and that is becoming 3 and accordingly the corresponding whenever I say print which one will be printed, which one will be printed will be the one that is within its scope this a is in the scope of this. So, that was printed this and gone this A was is in the scope of this. So, printed and gone and this A is in the scope of this and this is printed. So, this is known as the scope of variables ok.

(Refer Slide Time: 20:38)



Function: Summary

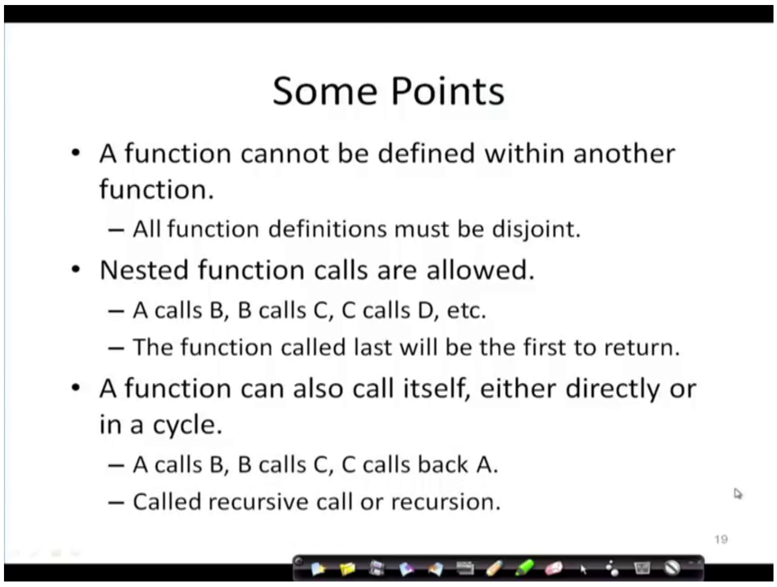
```
#include <stdio.h>
int factorial (int m)
{
    int i, temp=1;
    for (i=1; i<=m; i++)
        temp = temp * i;
    return (temp);
}
```

```
main()
{
    int n;
    for (n=1; n<=10; n++)
    printf ("%d! = %d \n", n,
    factorial (n) );
}
```

So, if we summarise functions, you can see this I do not know how much is visible. So, main function I am calling of a function factorial, and we have already seen that and then the function is having different its a self contained programme, which has got its definite name function definition whether type of the argument is also specified. Now main is a function and here I am calling a function, I am actually calling a function by name calling by name and is a returned data type repeating that the function name is there the parameter is there and the return statement.

And the other variables like temp and all those are local variables you are repeating.

(Refer Slide Time: 21:36)



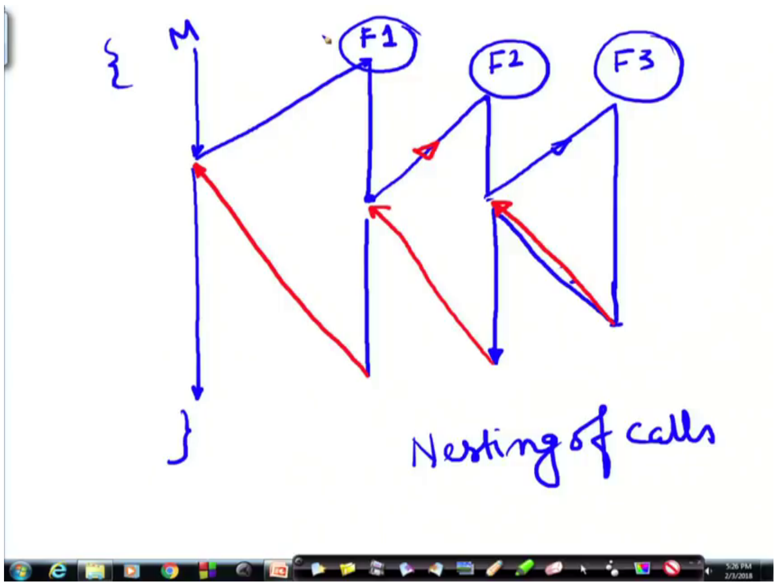
Some Points

- A function cannot be defined within another function.
  - All function definitions must be disjoint.
- Nested function calls are allowed.
  - A calls B, B calls C, C calls D, etc.
  - The function called last will be the first to return.
- A function can also call itself, either directly or in a cycle.
  - A calls B, B calls C, C calls back A.
  - Called recursive call or recursion.

19

Now some points is a function cannot be defined within another function, which we have told, but I am repeating it again all function definitions must be disjoint; that means, I cannot define one function within another nested function calls are allowed what is meant by nested function call nested function call means that suppose here is a main program going on.

(Refer Slide Time: 22:10)

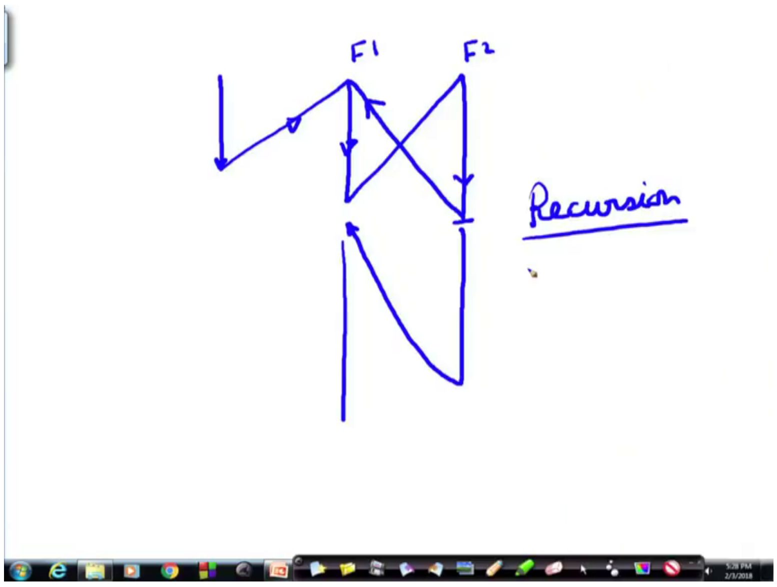


I call a function this is function say F 1, and this was my main from some point in the function in order to solve this problem F 1, I may call another function F 2 this is nesting calling, but not defined they are defined separately F 1 defined separately, F 2 defined separately, m defined separately. Now from F 2, I may call another function F 3 and F 3 completes F 2 has called F 3 because of some reason. So, that problem reason is answered I mean for some value of a some computation that computation has been done from here it returns to the point from where it was called. So, this a return point. So, is it clear or should I use some other caller for the return point is it necessary. So, I am not getting the colour. So, let me use the existing colour whatever was there. So, let it go. So, from here now the colour has come. So, I can show the return here I am returning, but then again I am continuing with this function.

And when F 2 is over, then I again return to F 1. So, it was called F 1 called F 2 for some purpose that purpose is solved. So, I return here and then F 1 continues again in its whatever it has doing and F 1 was called by main for some particular reason when that purpose is served, then we return to this point and then main continues right and ultimately main ends with this bracket. Now here this is known as nesting; that means, I nesting of calls.

So, I have made a call, and from that call I can make another call from there I can make another call. But the point to point to note is that all this functions must be independently and separately defined they cannot be defined one among the other. So, nested function calls are allowed A calls B, B calls C as I have shown m calls F 1, F 1 calls F 2, F 2 calls F 3 like that it can happen. The function called last will be the first to return; obviously, we have seen that in our earlier slide that we go back to the F 4 from F4 I return to F 2 and like that and. So, a function can call also call itself either directly or in a cycle, we will see this separately what is meant by that. A calls B this is this can be in 2 ways one is that say A calls B, B calls C, C calls back A that is possible like say here if we see that.

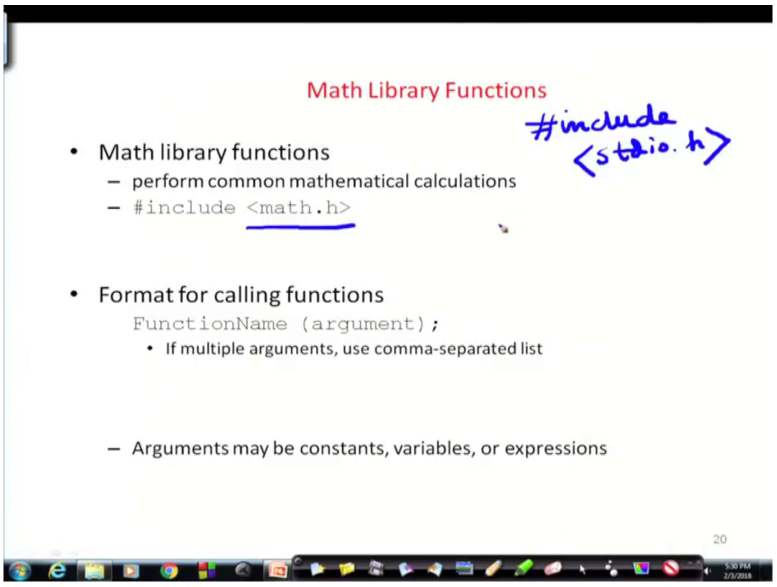
(Refer Slide Time: 26:14)



Main was running it called F 1, F 1 called F 2 and then F 2 can again call F 1 and then this call ultimately for this call the return has to come here and ultimately it will have to return here etcetera. So, this part will have to see separately that is its a function one function is calling another and that can be in a cycle F 1 calling F 2, F 2 calling F 1 it can happen or recursion means say a particular function F 1 calling itself a number of times that requires a special attention and a special discussion that we have to carry out we will do that in subsequently.

But right now just it is remember that these calls can be in a cycle or it can be called to itself which is a recursion.

(Refer Slide Time: 27:30)



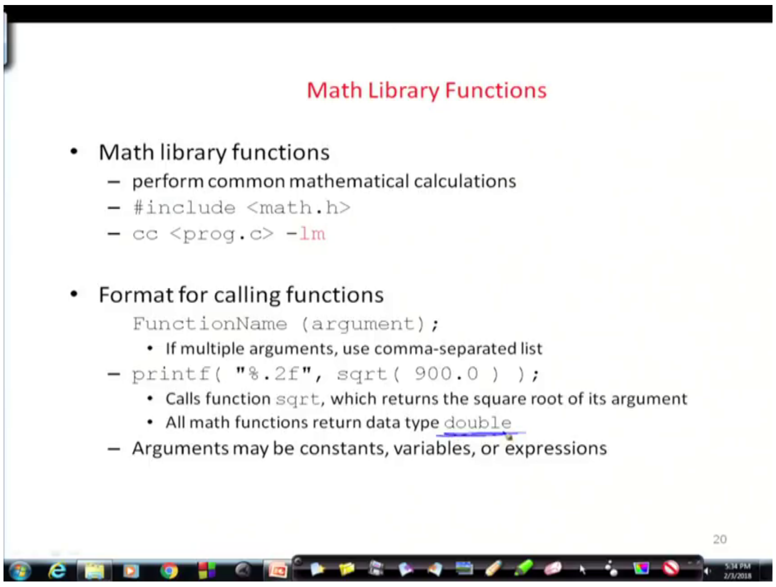
Math Library Functions

- Math library functions
  - perform common mathematical calculations
  - `#include <math.h>`
- Format for calling functions
  `FunctionName (argument);`
  - If multiple arguments, use comma-separated list
  - Arguments may be constants, variables, or expressions

20

Now, we have got some math library functions, which perform common mathematical calculations and I do not remember whether in an earlier class I mistakenly said that, I had mistakenly I do not remember exactly whether I did that or not that you need to include just as we include math dot s t d i o dot h, similarly we have to include math dot h. So, just as we include s t d i o dot h if we use some mathematical functions which are already available in the c library, we have to include math dot h. I do not remember whether while first introducing the square root function, I have might be mistakenly I wrote math dot leap that is a library function. So, dot leap if I had said that that you should ignore it is math dot h include math dot h and.

(Refer Slide Time: 29:11)



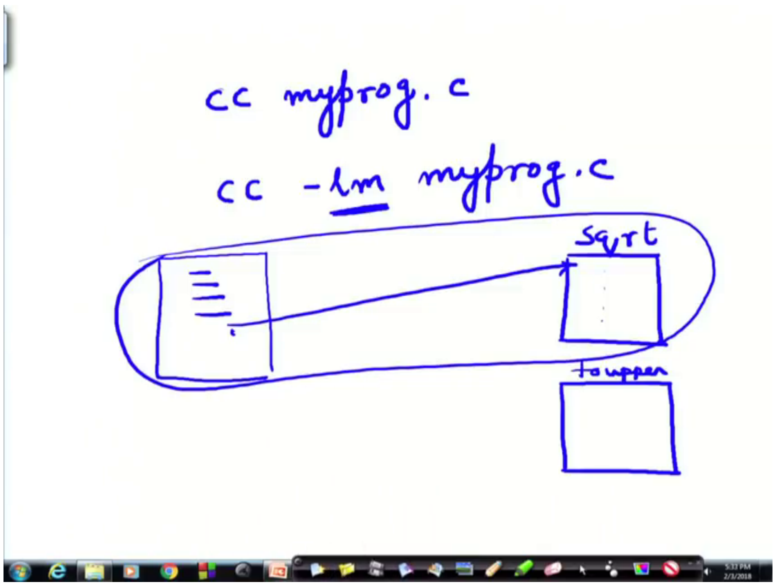
Math Library Functions

- Math library functions
  - perform common mathematical calculations
  - `#include <math.h>`
  - `cc <prog.c> -lm`
- Format for calling functions
  - `FunctionName (argument);`
    - If multiple arguments, use comma-separated list
  - `printf( "%.2f", sqrt( 900.0 ) );`
    - Calls function `sqrt`, which returns the square root of its argument
    - All math functions return data type `double`
  - Arguments may be constants, variables, or expressions

20

So, here there is an important thing, when I compile you known any function that we any program that we write we have to compile it in order to get and executable code. Now in your exercises you must have done by now the typically you compile a c program like this.

(Refer Slide Time: 29:33)



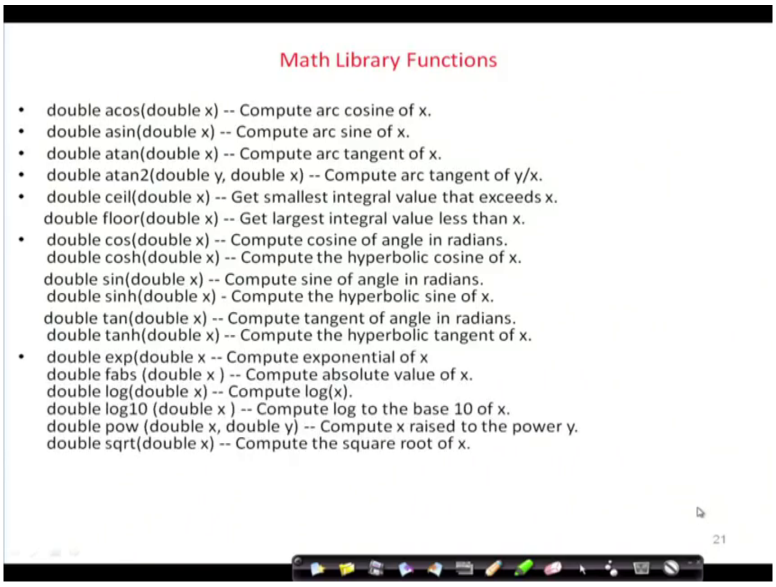
C c myprog dot c right, but if you use some mathematical library in your function then you should write c c minus l m myprog dot c or; that means, link to the mathematical library you compile first you compile now you see.

What is happening is the mathematical libraries are here say some square root function somebody has written for you and that is in the c library alright may be some other function like to upper, which converts from lower case to upper case all these things are there. Now to upper is the separate library where square root is the math library. So, if in your function you if in your program you write you refer to the square root some mathematical library, then you must do this why because purely myprog dot c will generate some object code alright object code.

Now the code for this has to be linked to this has to linked. So, that your ultimately the fool executable code also takes into this account this code link to this code will be forming your executable code because the square root you will need anyway at that time of running it right. So, here it has shown minus l m at the later also c c program name and then link with the mathematical library format for calling the functions, forget about this let us make it for the time being ignore this just say percentage f function name. So, this is point number 1, there are many mathematical library functions in order to include in order to use them we have to include immediately after s t d i o dot h hash include math dot h and we must use this give this linking command.

Now, when we call the function, the function name argument if multiple arguments then we can use a comma separated list say for example, printf some format the square root nine or there is only one. Argument not much arguments may be constant variables or expressions all math now this is important. All math functions return the data type double this is important you should keep in mind this are for c, all the math functions are returning the data type double. So, in order to make it compatible with the variable where you except the value returned by the math function that should also be double arguments may be constants or variables.

(Refer Slide Time: 33:22)



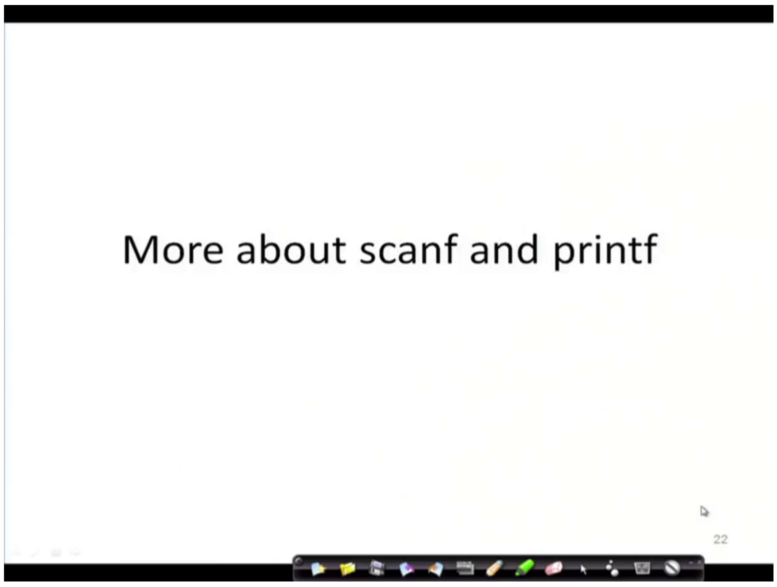
Math Library Functions

- double acos(double x) -- Compute arc cosine of x.
- double asin(double x) -- Compute arc sine of x.
- double atan(double x) -- Compute arc tangent of x.
- double atan2(double y, double x) -- Compute arc tangent of y/x.
- double ceil(double x) -- Get smallest integral value that exceeds x.
  double floor(double x) -- Get largest integral value less than x.
- double cos(double x) -- Compute cosine of angle in radians.
  double cosh(double x) -- Compute the hyperbolic cosine of x.
  double sin(double x) -- Compute sine of angle in radians.
  double sinh(double x) - Compute the hyperbolic sine of x.
  double tan(double x) -- Compute tangent of angle in radians.
  double tanh(double x) -- Compute the hyperbolic tangent of x.
- double exp(double x -- Compute exponential of x
  double fabs (double x) -- Compute absolute value of x.
  double log(double x) -- Compute log(x).
  double log10 (double x) -- Compute log to the base 10 of x.
  double pow (double x, double y) -- Compute x raised to the power y.
  double sqrt(double x) -- Compute the square root of x.

21

So, here are some examples of math library function like finding the cos of some angle x finding the sin thus all this functions are known as a sin a cos a tan inverse tan (Refer Time:33:41) tan ceiling function floor function. Floor function means it finds the greatest largest integral value that is less than x, suppose some say 200.56. So, the largest integral value is 2 hundred that is the floor function. So, cos a cos is finding the cos of an angle in degree whereas, a cos sorry cos is for finding the cosine of angle in radian now there is no point in memorising them as and when you need them look at the manual look at the book and very soon you will get a custom to the different library which are available for c and.

(Refer Slide Time: 34:35)


More about scanf and printf

22

We will come back to this in the next function, next lecture about some more very well-known functions which you have already encountered with and then we will proceed further with recursion.

Thank you.