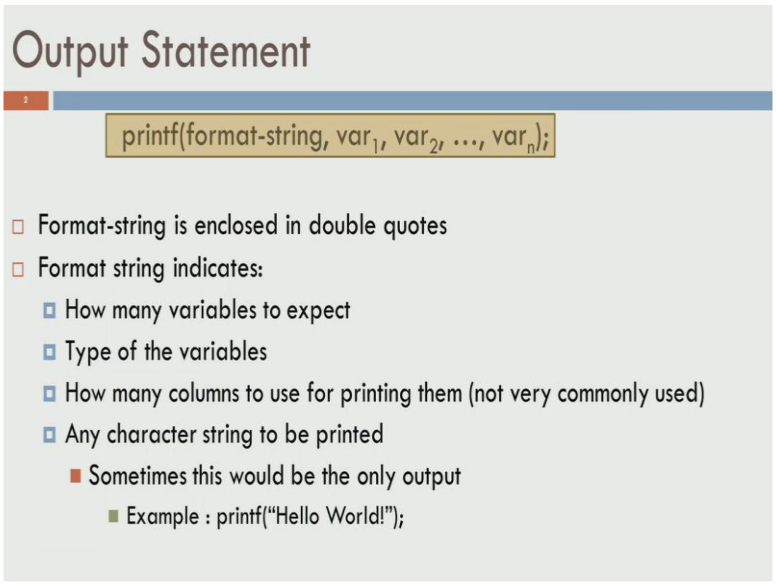
Programming, Data Structures and Algorithms

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

Lecture – 05
I/O statements
Printf, Scanf
Simple statements,
Compound statements

Welcome to lecture 2 of this online course on programming. In today’s lecture, in the first two modules we looked at input statements and output statements, compound statements and will also look at this category of statements called the selections statements.

(Refer Slide Time: 00:28)



Output Statement

printf(format-string, var_1, var_2, …, var_n);

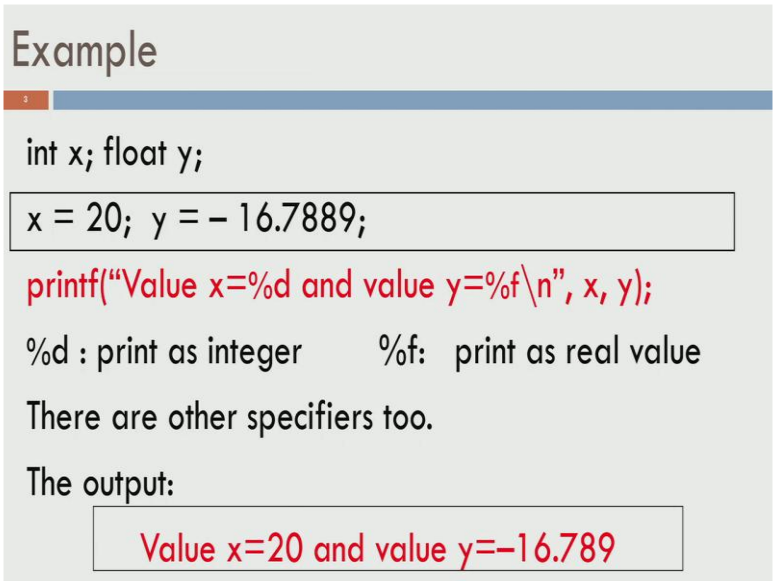
- Format-string is enclosed in double quotes
- Format string indicates:
  - How many variables to expect
  - Type of the variables
  - How many columns to use for printing them (not very commonly used)
  - Any character string to be printed
    - Sometimes this would be the only output
      - Example : printf(“Hello World!”);

So, we have been looking at printf and scanf for a while and I have not really explained what printf and scanf are about yet. But, in this lecture we look at what is the syntax of printf and what are the different things that go into the format itself. So, printf in a general form has a format string followed by list of comma separated variable names. So, the syntax is highlighted at the top, so you have format string comma, a comma separated list of variables 1 to n.

So, format string is enclosed in double quotes. So, we are seen that in several examples before, I will also show your examples later. And the format string essentially indicates how many variables should it be printing on the screen, what are the types of the variables, how many columns to use for printing them. So, numbers could occupy different number of columns and so on. So, we could have right justification or left justification and so on. So, we can specify the number of columns required, even though that is not something that you would see in this lecture in detail.

And if you want any character string to be printed, for instance the very first program that we wrote, we did not really have any variables to be printed, we just said printf hello world.

(Refer Slide Time: 01:53)



Example

```
int x; float y;
x = 20;  y = – 16.7889;
printf("Value x=%d and value y=%f\n", x, y);
```

%d : print as integer   %f: print as real value

There are other specifiers too.

The output:

```
Value x=20 and value y=-16.789
```

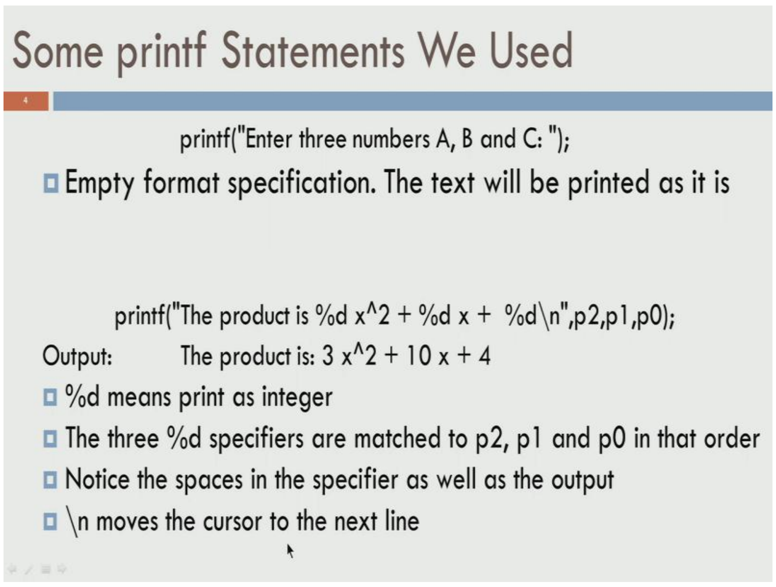
So, let us look at this example, let us say I have int x and float y and let us say x equals 20 and y is minus 16 points something. If you have this printf statement value x equals percentage d and value y equals percentage f back slash n comma x comma y. So, that is the print f statement, so as you can see the very first part as the format string which is enclosed to within double quotes and then there is a comma separator list of variables.

In this case it is a comma separator list of two variables namely x and y. And when you look at this one, there is string value space x symbol equal to and then the something called percentage d and then, there is some other string up to here percentage f back slash n. So, percentage d says that, whatever value it is or variables it is going to get, it should print it as integer and percentage f indicates that, whatever it is going to get as a variable it is supposed to be printed as a real value.

So, this percentage d is the very first percentage symbol, that you see followed by some letter d. So, this percentage d associate itself with the first variable x and the second one percentage f here associated itself with y here. And so since percentage d is for integer x is printed as integer and since percentage f is for real y is printed as real. So, this is the basic example, there are other specifiers, but more often than not you will need only d f and possibly a g.

So, the output for this would be value x equals 20 and value y equals minus 16.789. So, the idea behind is that, if you halve this strings here. So, this value appears as it is this space appears as it is x equals to appears as it is, only this percentage d is a place holder. So, it is holding it is place for this variable to be printed in its appropriate format.

(Refer Slide Time: 04:06)



Some printf Statements We Used

```
printf("Enter three numbers A, B and C: ");
```

- Empty format specification. The text will be printed as it is

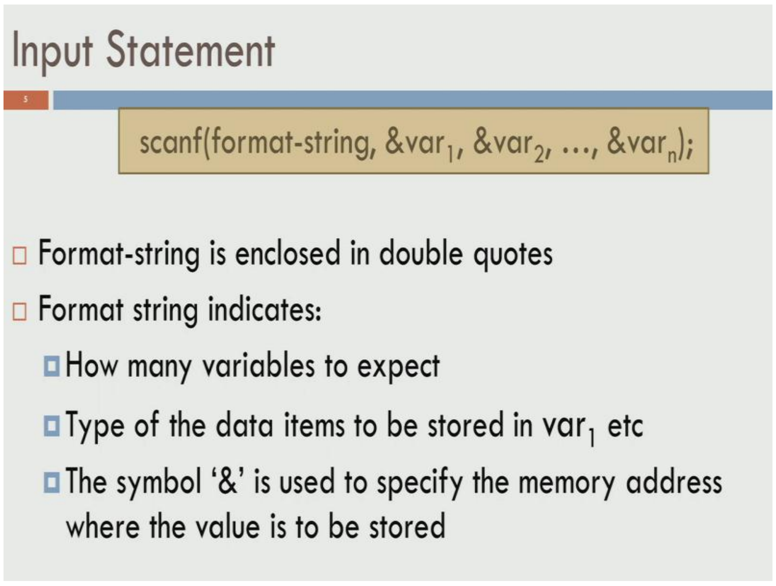
```
printf("The product is %d x^2 + %d x +  %d\n",p2,p1,p0);
Output:        The product is: 3 x^2 + 10 x + 4
```

- %d means print as integer
- The three %d specifiers are matched to p2, p1 and p0 in that order
- Notice the spaces in the specifier as well as the output
- \n moves the cursor to the next line

So, let us look at some of the printf statements we are use before, printf enter the three numbers A, B and C. So, this is something that we are used before. So, this is just a format specification, there is nothing to be printed at here, it will print as text as it is, then look at this other statement printf, the product is percentage d x char at 2 plus percentage d x plus percentage d back slash n. So, this is something that we saw in the polynomial multiplication example in the previous lecture.

So, now, this percentage d associate itself with p 2 this one with p 1 and the third one with p naught. So, the output that we got lost time was 3 x squared plus 10 x plus 4. So, you can see that percentage d was place holding for 3 and percentage d was place holding for 10 here and this one for 4 and everything else get printed as it is. So, back slash n as a mention earlier, it is essentially is printing a new line. So, it will print this statement, the product is this and the any other print that you give will go to the next line.

(Refer Slide Time: 05:18)



Input Statement

```
scanf(format-string, &var1, &var2, ..., &varn);
```

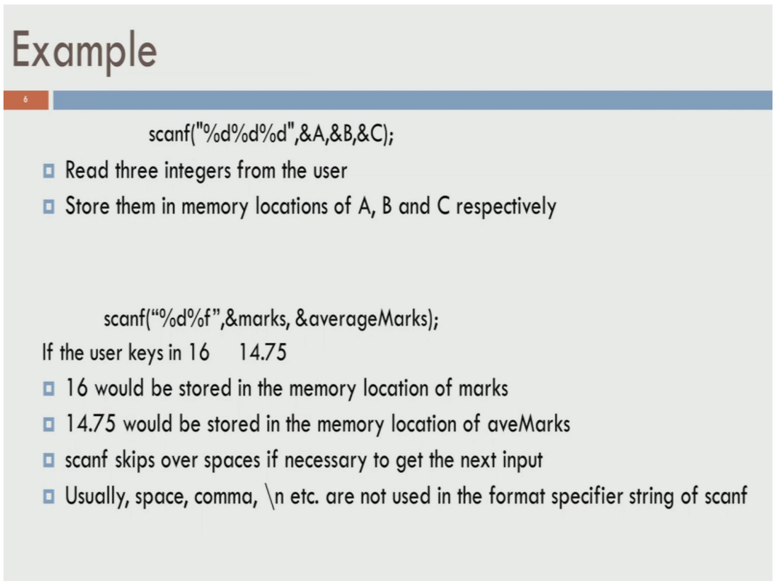
- Format-string is enclosed in double quotes
- Format string indicates:
    - How many variables to expect
    - Type of the data items to be stored in var_1 etc
    - The symbol '&' is used to specify the memory address where the value is to be stored

What about the input statements? So, in the input statement let us look at scanf. So, scanf is also of the similar format, you have format string and you have a comma separated list of variables that you want to be scanning. So, the format string as before is enclosed in double quotes. And the format string indicates, how many variables to expect as it was in printf, type of the data items should be stored in var 1, var 2 and so on.

So, it is very similar to printf except that you see the symbol ampersand in front of all the variables. So, you have ampersand var 1, ampersand var 2 ampersand var n and so on. So, the symbol ampersand is use to specify the memory address, where the value is to be stored. So, remember var 1, whenever you use a directly in your program, it goes to the memory location and gets the value. But, in this case scanf is supposed to take the value from the user and put it in the memory location called var 1.

So, it do that you put ampersand of var 1, we look at this in more detail later, what you are essentially passing onto scanf is a pointer to var 1 by prefixing and ampersand before it. So, ampersand var 1 is a pointer to var 1 and it says, this is the memory address for var 1 instead of calling it by name, you now get the memory address and whatever the user inputs will go to that memory address. So, we saw this example in the previous lecture also, when scanf happened we went to the specific location and we got the values for a, b, c, d and so on.

(Refer Slide Time: 07:04)



Example

```
scanf("%d%d%d",&A,&B,&C);
```

- Read three integers from the user
- Store them in memory locations of A, B and C respectively

```
scanf("%d%f",&marks, &averageMarks);
```

If the user keys in 16   14.75

- 16 would be stored in the memory location of marks
- 14.75 would be stored in the memory location of aveMarks
- scanf skips over spaces if necessary to get the next input
- Usually, space, comma, \n etc. are not used in the format specifier string of scanf

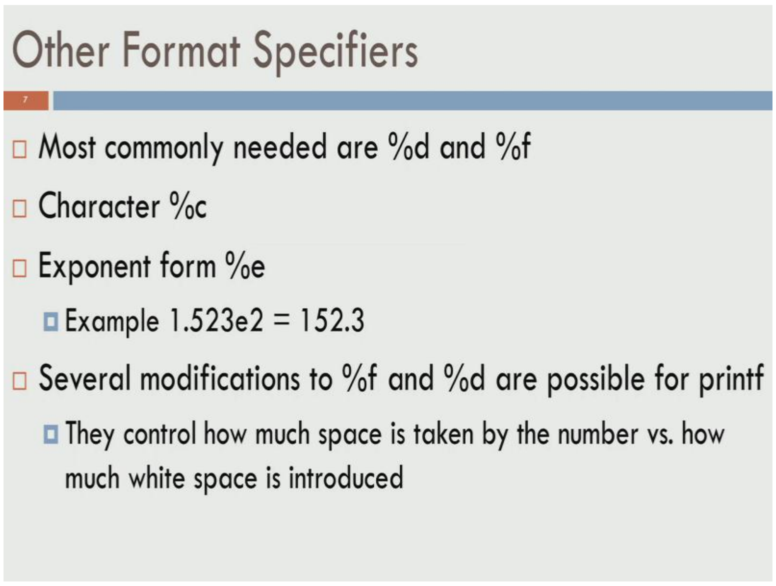
So, let us look at the other example, we have seen before, scanf percentage d percentage d percentage d ampersand A comma ampersand B comma ampersand C. So, again the order in which things are done is left to right for the format specifiers, as well as left to right for the variables here. So, it will expect three integer from the user and put these three in locations A, B and C respectively. Finally, if we have this kind of an example, where percentage d percentage f comma ampersand marks ampersand average marks.

So, I am assuming that marks is an integer and average marks is a floating point value. So, if the user keys in 16 let say some couple of spaces and 14.75. First the scanner will come and look at it from the left side and keeps scanning tell it finds in integer, in this case 16 is in integer and after that it is a whitespace, whitespace is not a part of an integer. So, it will stop scanning, so there is a number 16 which the valid integer, that goes into the very first variable namely marks.

Then, the format specifier says percentage f which means, the scanner should look for a floating point number. And when it looks at this spaces 1, 2, 3, 4 spaces the spaces not part of a number. Therefore, it keeps looking further it sees number 14, but there is a dot and followed by 75 after that there is no more input, which means the input ends at 14.75, which is 14.75. So, 14.75 would go as the average marks.

So, usually space, comma, back slash n and so on or not specified in the format specifier, in the format specifier which put within double quotes, you usually do not put space comma and other things. In printf it is useful to print something on the screen, in scanf you just want the data from the user and the user may enter space in sonic and scanf automatically ignore set. So, usually you do not put it as part of the format specifier. So, this scanf skips over space is necessary to get the next input.

(Refer Slide Time: 09:16)



Other Format Specifiers

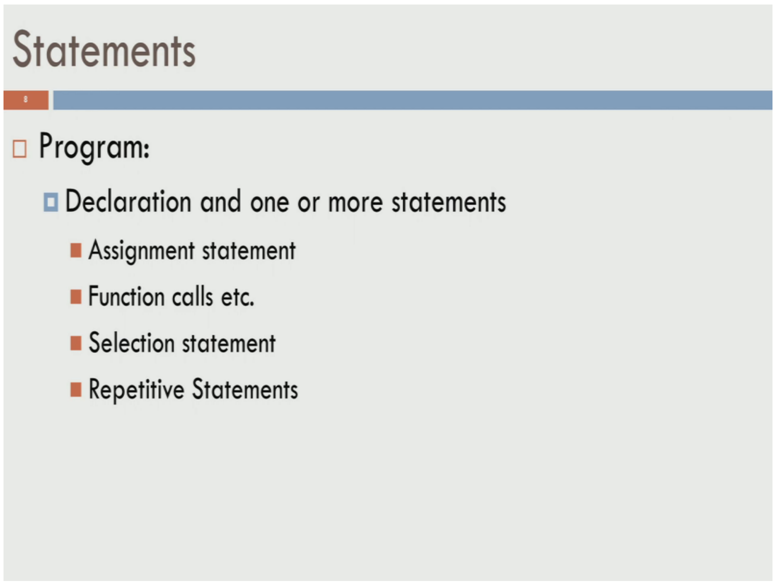
- Most commonly needed are %d and %f
- Character %c
- Exponent form %e
  - Example 1.523e2 = 152.3
- Several modifications to %f and %d are possible for printf
  - They control how much space is taken by the number vs. how much white space is introduced

There are several other the format specifier that you can use. But, the most commonly needed are percentage d and percentage f, the other one that might come useful is percentage c, where c is transfer character. So, percent percentage d is decimal, percentage f is float, percentage c is character. There is also an exponent form for floating point numbers, which goes by percentage e. For example, 1.523 e 2 means 1.523 into 10's square or 10 power 2 and which is 152.3.

So, there are also several modification to percentage f and percentage d which are possible, which you can use in print f. However, they only control how much space the number takes on the screen, it does not really control how the value is itself. So, if you say percentage d it is an integer, so it will get printed as an integer. But, depending on percentage followed by a number and a d will take as many spaces as specify the number, this is useful and justification and so on.

So, I am not going to cover how to have a fixed width of numbers and so on. So, that is beyond the scope of this lecture. So, I suggest that it you go and read up a book on C to get more details.

(Refer Slide Time: 10:39)

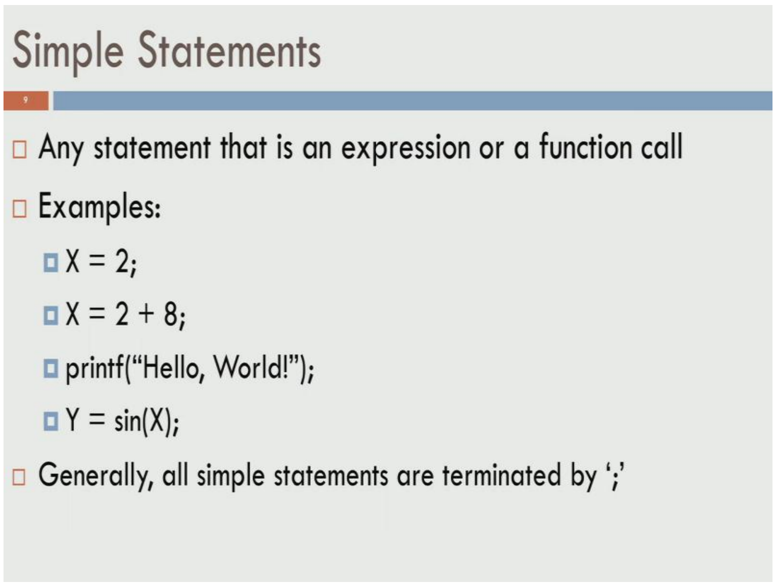


Statements

- Program:
  - Declaration and one or more statements
    - Assignment statement
    - Function calls etc.
    - Selection statement
    - Repetitive Statements

Let us move on to statements, so we know that a program is usually a declaration followed by one or more statements. And the statements could be an assignments statement or function calls, we have already seen these two kinds. So, assignment statements we are seen them in the polynomial example and function calls printf and scanf for functions, we have seen them also, we saw selection statement also when we looked at maximum of three numbers and so on, one thing that we have not seen is repetitive statements. So, statements could be of four types, assignment, function calls, selection statement and repetitive statements. The first two are of a category, and the second two are of another category, which is what I want to talk about in the next few slides.

(Refer Slide Time: 11:21)



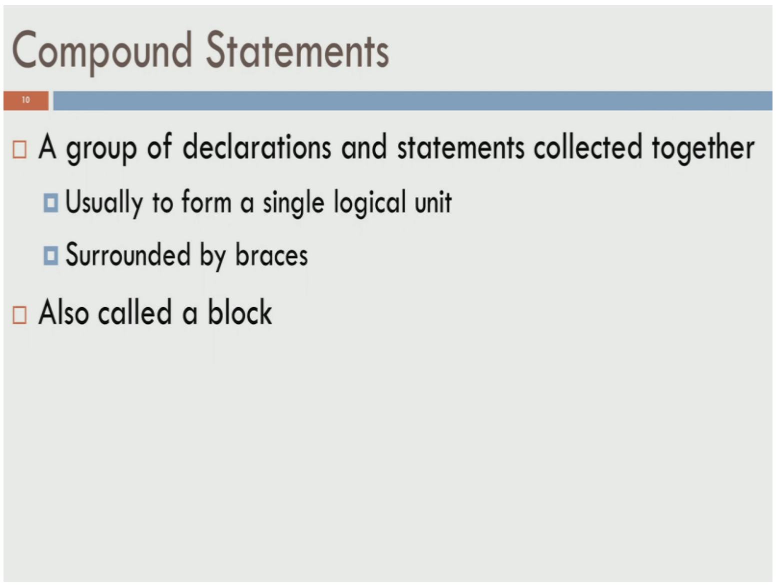
Simple Statements

- Any statement that is an expression or a function call
- Examples:
  - X = 2;
  - X = 2 + 8;
  - printf("Hello, World!");
  - Y = sin(X);
- Generally, all simple statements are terminated by ';'

So, the first category we call simple statements, are those statements that are expressions or function calls. So, for example if I have x equals 2, x equals 2 is single statement you take the value to assigned two x. So, it is an assignment statement. Similarly x equals 2 plus 8 it compute 2 plus 8 on the right hand side gets the value 10 puts it to the left hand side, it is a simple statement. Because, it is doing only an assignment, then there is a function call printf hello world, so it is a simple function call.

And finally, sin of x, so x is passed as a parameter to function call sin, sin computes that and returns the value which is placed in y. So, these are simple statements, because the right hand side evaluates the value and the value is put into the variable on the left hand side. So, simple statements are usually terminated by a semicolon and in this case, you can see that all these examples all this four different simple statements have a semicolon at the end. So, anywhere were you evaluate and expression and so on will all be simple statements, anywhere where you call functions will all be simple statements as well.

(Refer Slide Time: 12:38)



Compound Statements

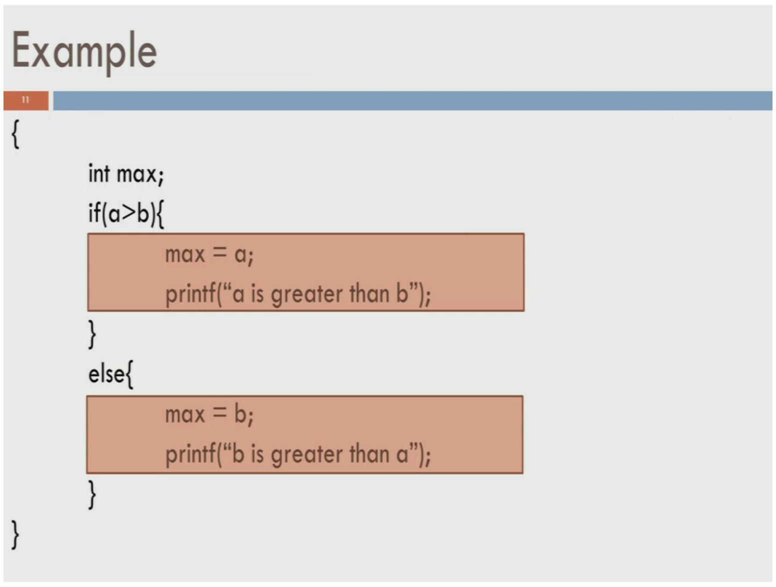
10

- A group of declarations and statements collected together
  - Usually to form a single logical unit
  - Surrounded by braces
- Also called a block

So, the other category is what is called compound statements, a compound statement is a group of declarations and statements collected together. So, compound statement contains one or more set of simple statements and it may also have declarations on there own. So, for example, a single simple statement is also a compound statements because, compound statement contains one or more simple statements.

So, the basic idea begin compound statements is that, it forms one logical unit and to say that this is one logical unit we surround the statements that are grouped together using braces. So, sometimes the set of statements that are within the braces are also called a block, so let see these examples.

(Refer Slide Time: 13:27)



Example

```
{
        int max;
        if(a>b){
                max = a;
                printf("a is greater than b");
        }
        else{
                max = b;
                printf("b is greater than a");
        }
}
```

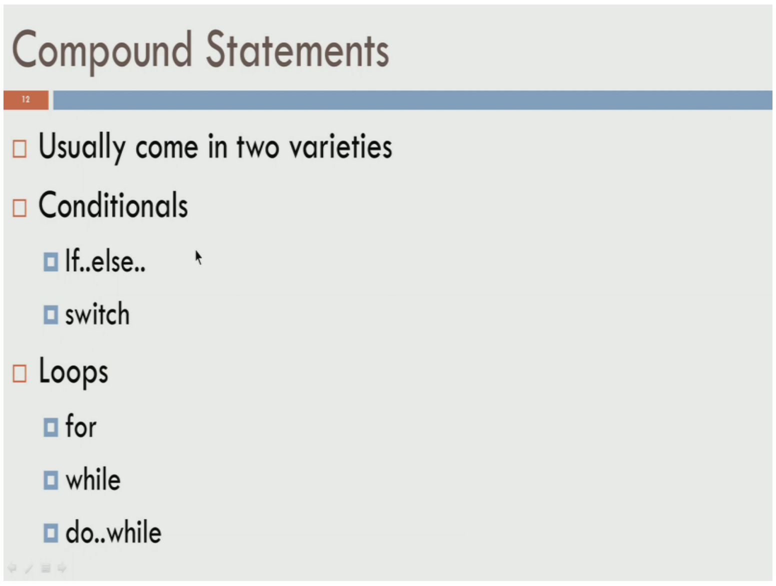
So, we have seen something like this, let say I want to calculate the maximum of two numbers. So, I have a small program segment which does that, so you have int max if a is greater than d max is a and printf a is greater than b, else max is b and printf b is greater than a. So, we are not only printing whether a is greater than d or b is greater than a, we are also storing the maximum value in this variable call max.

So, if you notice you have braces left and right braces for, if a greater than b and we have left and right braces for the else clause also and this is one logical unit. So, if a is greater than b the maximum is indeed a and you can also print. So, this one logical unit, so I highlight it in orange here; however, the compiler only we look at this left and right bracket, anything with in this left and right brace is a block and this is treated as everything that should go under the true case.

If a is greater than b is false, then max is equal to b and you can print that b is greater than a. So, in fact b is greater than or equal to a in this case, it is not just greater it is also greater than or equal to. So, this is one logical unit and the braces say that this is one logical unit. So, this is a compound statement and this whole if statement by itself is a compound statement, because if statement… So, this if followed by an expression followed by a block else followed by a block, this whole thing is also one logical unit therefore, it is also a compound statement.

So, compound statement may have one or more compound statements inside them and it may also have simple statements inside them. A simple statement is one line expression or a function call and so on.

(Refer Slide Time: 15:27)



Compound Statements

12

- Usually come in two varieties
- Conditionals
  - If..else..
  - switch
- Loops
  - for
  - while
  - do..while

So, compound statements in turn can come in two varieties, they can either be conditional statements namely, if then else statements or it could be switch statements. The other variety is also called the repetitive statements or loops I refer to them earlier as repetitive statements. So, there are three varieties of loops available in C namely, for loop, while loop and do while loop. So, we will look at more details in the next few slides about, if then else statements and switch statements and later in the lecture you will see things about loop statements.

So, for in this module what we are done is, we look that I/O statements and we look at two categories name we, simple and compound statements. Simple statements we already know the involve arithmetic expressions or function calls on the right side in variable assignment on the left side, compound statements is a collection of either compound statements or simple statements.