

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

**Lecture – 13**  
**Logical Operators and Change in Control Flow**

(Refer Slide Time: 00:21)

**Logical Operators**

- There are two logical operators in C (also called logical connectives).
  - && → Logical AND
  - || → Logical OR
- What they do?
  - They act upon operands that are themselves logical expressions.

*Temp > 50 — ①*  
*a\*b+c >= 25 — ②*

*Temp > 50    &&    a\*b+c >= 25*  
*∅1        &&        1 ⇒ ∅1*

52

In the last lecture we had talked about two different types of operators, relational operator and arithmetic operators. Today we will be discussing about the third type of operator which is known as a logical operator.

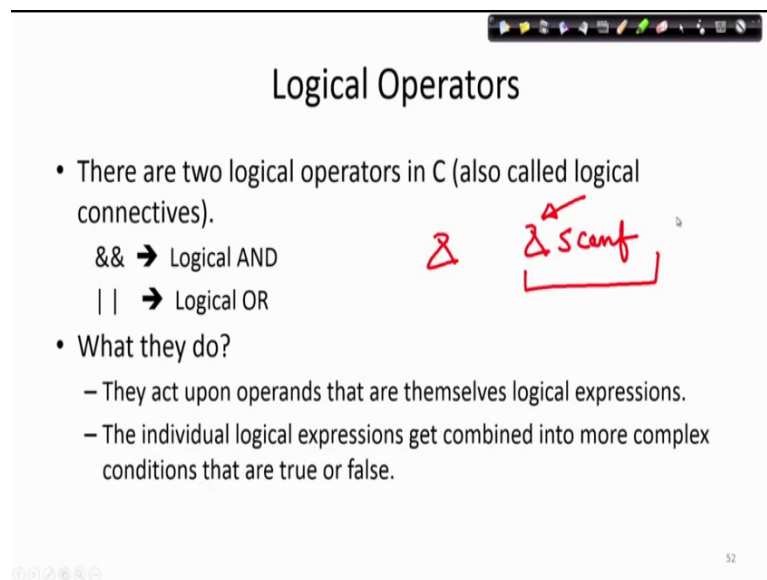
Logical operators are also known as logical connectives. So, there are two essentially there are two logical operators in C, one is logical AND and the other one is a logical OR. Now, what do they do? They act upon the operands themselves which are logical expressions. For example, let us say I am writing a logical expression temperature is greater than 50, now this will this statement suppose the temperature now is 40 degree centigrade then temperature greater than 40 greater than 50 will result in false value, because the relational operator will always generate either true or false. And suppose there is another logical expression a times b plus c is greater than equal to 25, now this is another logical expression sorry this is another relational expression. On the left hand side of this expression I have got an arithmetic expression and on the right hand side I have got a constant and I am connecting them with a relational operator greater than

equal to. Now, if a times b plus c is greater than 25 or equal to 25 then this will result in true.

Now, I can connect these two this one and this two together and write another expression like temperature greater than 50 and a times b plus c is greater than equal to 25. Now, this expression is a combination of two relational expressions and a logical operator a logical connective. This logical AND means that this entire thing expression will be true or will result in a 1 if both of them are true. So, if the temperature is 40 then this will become false or 0 and if this is 25 then this is true, but 0 and 1 both are not true therefore, this 0 and 1 will result in 0. But suppose if the temperature was 50 and a times b plus c is equal to 25 or greater than 25 then this is true and also this is true in that case these two together and because they are ANDed then this will be true.

So, the logical AND operator what it does is it turns a true value or 1 if all the components of the expression logical expression connected by the AND operator, logical AND operator is true.

(Refer Slide Time: 05:37)



The slide is titled "Logical Operators" and contains the following text:

- There are two logical operators in C (also called logical connectives).
  - && → Logical AND
  - | | → Logical OR
- What they do?
  - They act upon operands that are themselves logical expressions.
  - The individual logical expressions get combined into more complex conditions that are true or false.

Handwritten notes in red ink include an ampersand symbol (&) and the expression "&scanf" with an arrow pointing to the ampersand and a bracket underneath.

Now, one thing you can note here that since this ampersand is a character and we have already use this ampersand in expressions like AND scanf and we have discussed that this AND actually means we are trying to get the address of a particular I am sorry I am sorry absolutely sorry.

(Refer Slide Time: 06:10)

### Logical Operators

- There are two logical operators in C (also called logical connectives).
  - && → Logical AND
  - || → Logical OR
- What they do?
  - They act upon operands that are themselves logical expressions.
  - The individual logical expressions get combined into more complex conditions that are true or false.

*scanf("%d", &velocity)*

I actually what I write is scanf etcetera percentage d and velocity say where velocity is a variable alright. So, this AND in that case you is used to mean the address of this variable velocity.

So, in order to differentiate between this usage of AND and the logical operator logical AND is denoted as two ampersands, two ANDs.

(Refer Slide Time: 07:16)

### Logical Operators

- There are two logical operators in C (also called logical connectives).
  - && → Logical AND
  - || → Logical OR
- What they do?
  - They act upon operands that are themselves logical expressions.
  - The individual logical expressions get combined into more complex conditions that are true or false.

*exp1 || exp2 || exp3*

*exp1 && exp2 && exp3*

Similarly logical OR means that some expressions say, I have I write it in an abstract way say expression 1 or expression 2 or expression 3. Now, this composite expression

will be true if any one of them either expression 1 or expression 2 or expression 3 is true. If any one of them is true then this entire thing will be true, if two of them are true then also it will return true, if all the three are true then also it will be true, but if none of them are true if none of them are true then it will not be true, then none of them will be true.

So, what is the difference between this logical OR and logical AND therefore? In logical AND if instead of this it was written like if, instead of this it was written like expression 1 and expression 2 and expression 3 this composite and expression would be true only if all these three expressions are true alright. So, that is logical OR.

Now, what do they do? They act up on the operands that are themselves logical expressions why logical expressions where from did I get logical expressions I got the logical expressions from relational operators.

(Refer Slide Time: 09:32)

The slide is titled "Logical Operators" and contains the following text:

- There are two logical operators in C (also called logical connectives).
  - && → Logical AND
  - || → Logical OR
- What they do?
  - They act upon operands that are themselves logical expressions.
  - The individual logical expressions get combined into more complex conditions that are true or false.

Handwritten in red on the slide is the expression `t > 20` enclosed in a box, with an arrow pointing to it from below and another arrow pointing from the box to the text `1/0`.

A small video inset in the bottom right corner shows a man speaking.

For example, now I am writing some time is greater than 20 is a logical expression. What is this? This is a relational operator, but this expression is a logical expression. Why it is a logical expression? Because this will return only true or false nothing in between, so t greater than 20 if t is time or whatever value t might be if that is greater than 20 then it will return 1 or it will return 0. So, the logical connectives or the logical operators they are acting upon the operands themselves and connecting them. The individual logical expressions get combined into a more complex condition that are either true or false. We will see some examples.

(Refer Slide Time: 10:31)

- Logical AND


- Result is true if both the operands are true. *if all operands are true*

- Logical OR

- Result is true if at least one of the operands are true.

X	Y	X && Y	X    Y
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

*(Note: In the original image, the 'TRUE' in the bottom row of the 'X && Y' column is circled in red.)*



So, logical AND the result is true if both the operands are true had or for two operands. If a three operands if all the operands are true, it should be connected as if all operands are true and logical OR the result is true if at least one of the operands are true, if at least one this is most important.


So, let us look at the truth table here X and Y any of them can have the value false or true. Accordingly we can have four combinations X false, Y false, X false, Y true, X true Y false and both X and Y are true. Now, if I carry out the logical AND then for all these cases say false false the result will be false, so 0. False and one true still it will be false because here I want all operands should be true. One is X is true Y is false the result will be false if both of them are true the result will be true. While in the case of OR X or Y will result in false if X is false and Y is false, but if X is false and Y is true will get a true because I am interested in getting at least one to be true if this is true and this is false then also true if both of them are true then also it is true.

So, I think it is clear to you what is meant by the logical operators and how we can combine logical expressions based on that.

(Refer Slide Time: 12:45)

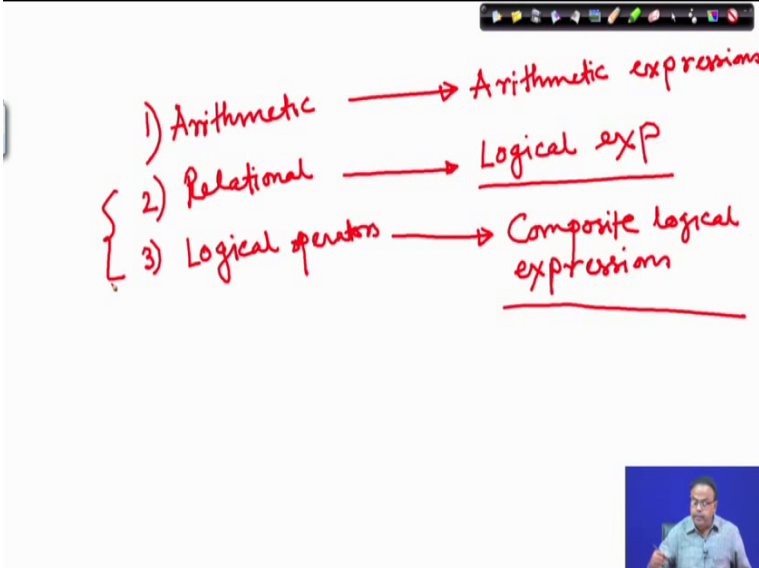
## Input / Output

- printf
  - Performs output to the standard output device (typically defined to be the screen).
  - It requires a format string in which we can specify:
    - The text to be printed out.
    - Specifications on how to print the values.  
`printf("The number is %d\n", num);`  
*The number is*          *ent*
    - The format specification %d causes the value listed after the format string to be embedded in the output as a decimal number in place of %d.



Now, we have seen three types of operations.


(Refer Slide Time: 12:55)



1) Arithmetic → Arithmetic expressions

2) Relational → Logical exp

3) Logical operators → Composite logical expressions

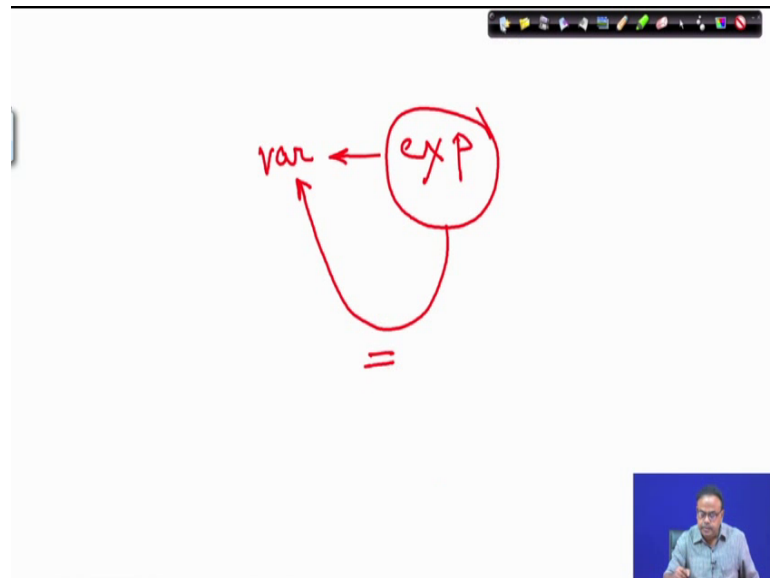


So, actually operators, one is the arithmetic operators, the next is relational operators and the third one is logical operators. Now, arithmetic operations use of arithmetic operations lead to arithmetic expressions. The use of relational operators lead to logical expressions why logical expressions, they lead to true or false value right logical expressions and use of logical operators will combine and get more complex logical expressions complex or let me not write complex, let me write composite that communicates the meaning better

composite logical expressions, composite logical expressions. So, will see the use of this pretty soon when we will be looking at the control operators right.

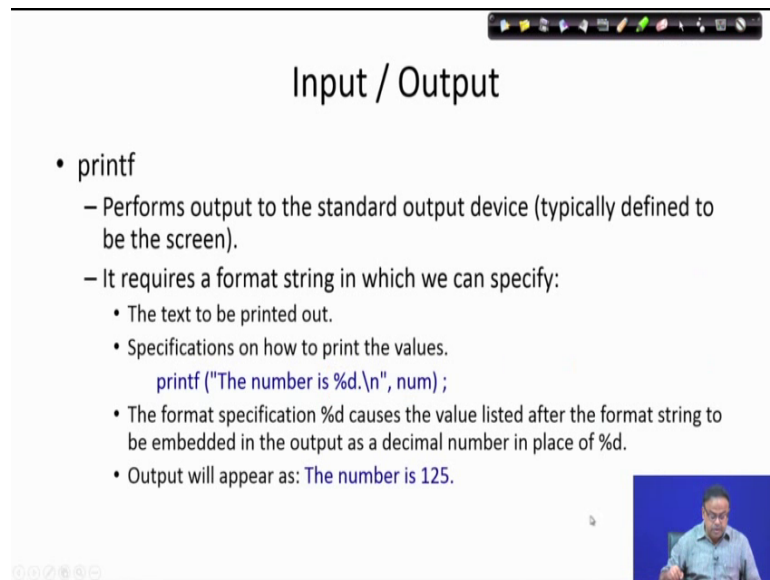
Next, just to wrap up the things let us come to the input output statements.

(Refer Slide Time: 15:46)



By the way besides this besides the arithmetic expressions, logical expressions we had seen another type of expression those are assignment expressions or assignment operators. By that means, the left hand side is an expression and right hand side is a variable we assigned the result of the computation of the right hand sorry I just said the opposite the right hand side is an expression on this side is an expression. Here on the left side is a variable and we compute the expression and assign the value of that computation to this variable and this is the assignment operator.

(Refer Slide Time: 16:13)



## Input / Output

- `printf`
  - Performs output to the standard output device (typically defined to be the screen).
  - It requires a format string in which we can specify:
    - The text to be printed out.
    - Specifications on how to print the values.

```
printf("The number is %d.\n", num);
```
    - The format specification `%d` causes the value listed after the format string to be embedded in the output as a decimal number in place of `%d`.
    - Output will appear as: `The number is 125.`

Now, besides that we have seen two other statements one is a `printf` statement we have seen that performs the output to the standard output device typically when we declare `stdout.h` then by default it is taken as a screen. The other one and it requires a format string in which we can specify the text we printed out and the specifications on how to print the values like `printf number is dash` and that dash can be filled up by percentage `d` for; that means, the specification is that an integer can come here. And then you remember what this means this means, I am going to the new line alright and then followed by the number. The format specification causes the value listed to be embedded here I have discussed that that you can consider this format to be a place holder alright, the number is dash and how can this dash be filled out the dash since its percentage `d` some integer value can come and fill it up right we have seen that.

The other statement that we saw is `scanf` that is for reading the values.



(Refer Slide Time: 17:41)

- scanf
  - Performs input from the standard input device, which is the keyboard by default.
  - It requires a format string and a list of variables into which the value received from the input device will be stored.
  - It is required to put an ampersand (&) before the names of the variables.

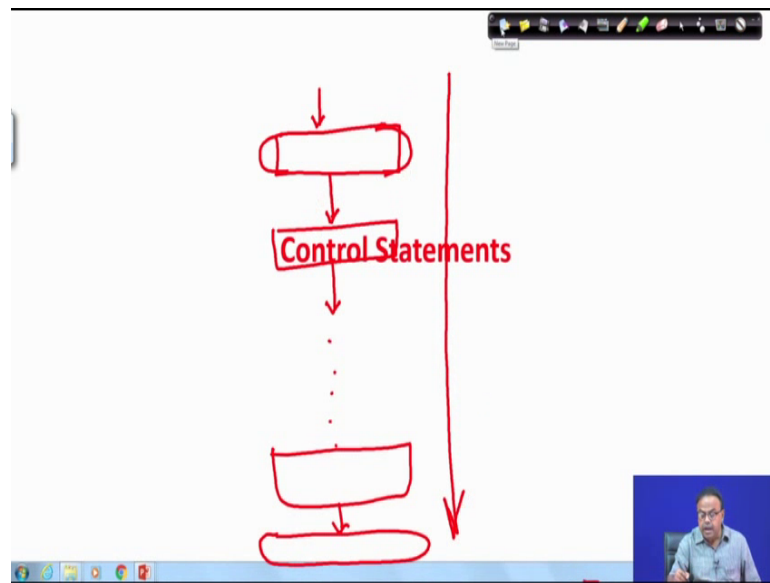
```
scanf ("%d" &size);  
scanf ("%c", &nextchar);  
scanf ("%f", &length);  
scanf ("%d %d", &a, &b);
```

So, it performs input from standard in input device, normally by default it is a keyboard and then it also requires a format string and list of variables like it is required to put an ampersand before the names of the variables, we have also explained why that is so. The reason is that this ampersand essentially means the address of that variable where the value that is being read will be put.

So, here are some examples, scanf percentage d and size; that means, what that I am reading in a variable size, size is a name of a variable and in which I am putting in some integer value alright. Similarly next char say is a say this one is a character variable. So, I am specifying that only a character can come in here and that is why I have put in the specification percentage C. Percentage f means some floating point number will come here, but in all these cases this ampersand means the address of the corresponding variables alright. Here percentage d, percentage d means sequentially I am going to read two integer variables a and b. So, all these we have seen and you will be best learning this by practicing it time and again and we will see a number of examples and in this course there will be quite a few assignments which you will have to do.

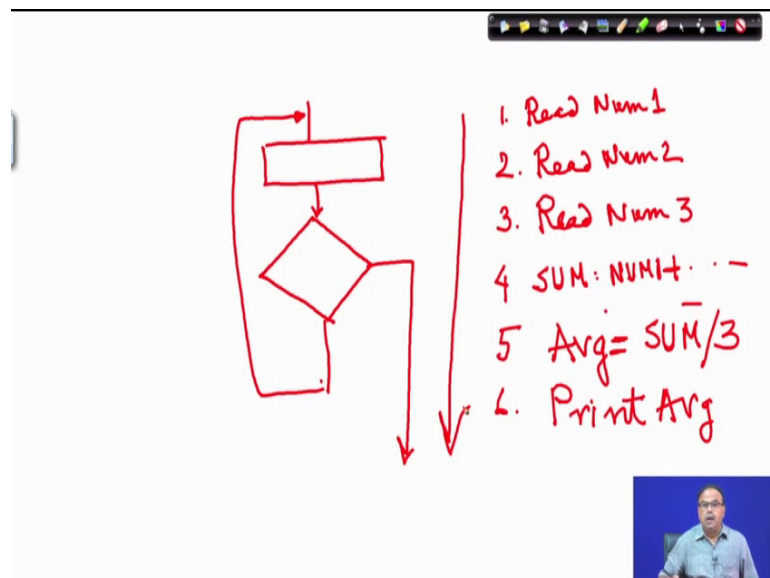
Next, we will move to, next we will move to a new topic which is a control structures and control statements. We have seen if we recall in a flowchart, let us go back to the flowchart where we have got some computation statements where we are doing some computations.

(Refer Slide Time: 20:27)



And we usually carry out one statement after another right that is how we do and in that way we will go on till the end of the program. For example, read number one, read number two etcetera divide add the numbers and divide the numbers to get the average. So, when we computed the average it was something like this and at the end we did some printf and in the meanwhile they were some reading the numbers these were some of them were input some were computations right. But it was a complete sequential thing.

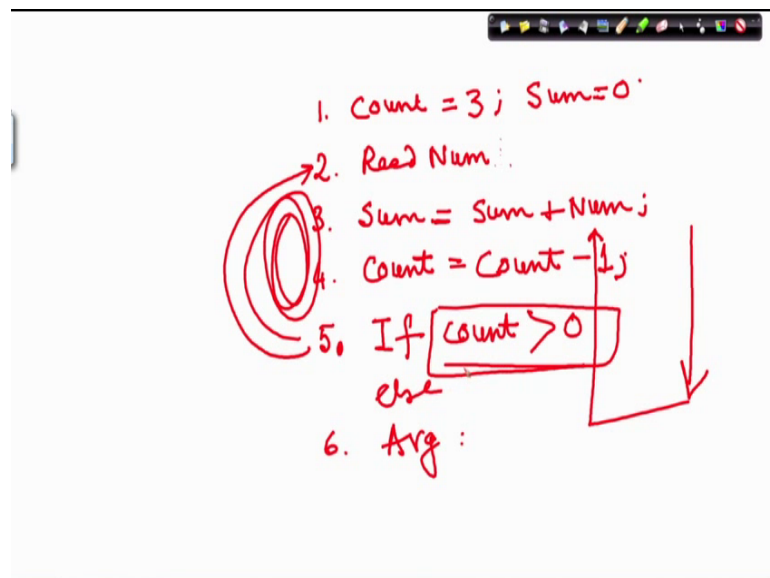
(Refer Slide Time: 21:45)



But if you recall the other type of, the other type of flowcharts that we had seen there we had started we did some computation and then we took some decisions and based on the decisions I have sometimes gone back to the earlier operation that earlier thing I had done and otherwise I would have followed this path. Typically in the examples that I we had sees we just see the pseudo code if I write say for computing the average of three numbers I will be read num 1, read num 2, read num 3 and then sum equals to num 1 plus num 2 like that I add them and then I compute the average to be sum divided by 3 right. So, and then we print the sum print the average.

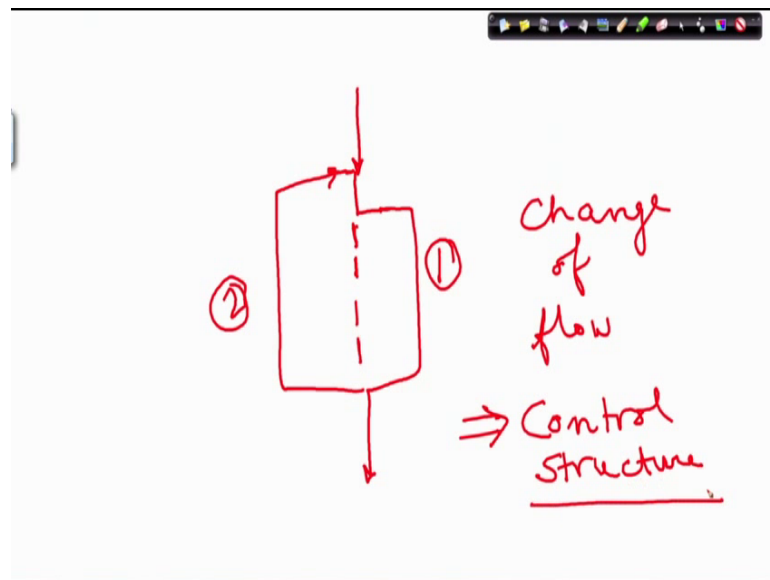
So, this entire thing is going in a sequence alright. Just one after another one after another no change in the path.

(Refer Slide Time: 23:37)



But in this case for example when we try to if you recall when we are not writing the this program in this form of pseudo code instead I initialise a count to be 3, then read number 1, sum and initially count here and say sum was 0, assign 0 and then sum is let me just make it make it just num I am reading one number and sum plus num and then count I decrement. So, I may count to be count minus 1 because I have, one I have already read. Now, I check if count is greater than 0 then what I do in my flowchart, I go back to reading the number again. Otherwise, else I come to the computation of the average right. Otherwise, if as long as count is greater than 0 I am continuously doing this thing right this steps.

(Refer Slide Time: 26:11)



So, here at this point you check that I am looking at the value of count and depending on the value of count I am deciding whether I will be going in this direction or I will go back and change the direction of the flow. So that means, as if the at this point the execution is undergoing a decision making to decide which part it will go through this path or this path.

As if you are therefore, you are controlling the flow of the program either in this way or you change the path, we will see that we can change the path in two different ways. One is that sequentially I am coming and I may go ahead I can skip some of the operations and I can go jump forward that is also a change of the path. This dotted line is showing the normal sequential execution or could be that some here I can go back to another path. So, these are two types of change of flow, change of flow that is that can be resulted in and that can be resulted in using the control structure.

So, we will soon go into the details of the control structure and see how such control structures or change of the sequential flow can be achieved in any programming will exemplify as we are doing for all others examples using the constructs of C, but that does not mean that it is restricted only to C. It can be, there are similar control constructs for other languages as well. We will come to that in the next lecture.