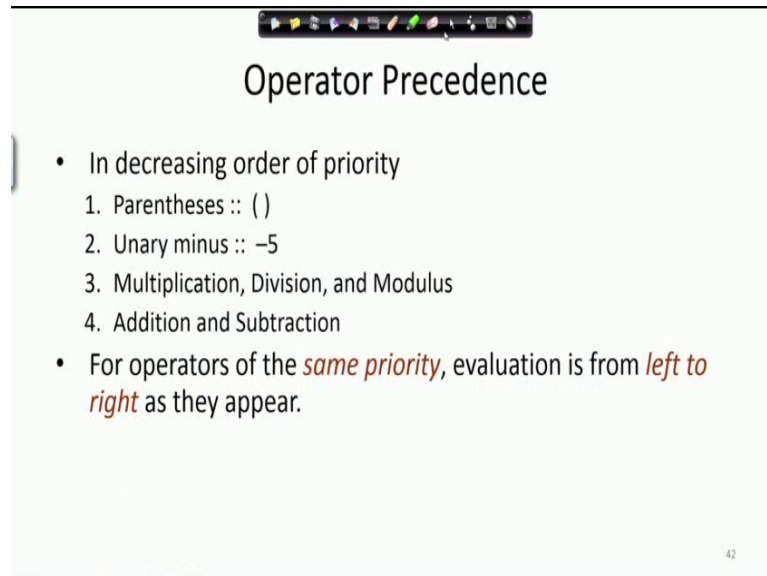


Problem Solving through Programming In C
Prof. Anupam Basu
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture – 12
Arithmetic Expressions and Relational Expressions

(Refer Slide Time: 00:20)



Operator Precedence

- In decreasing order of priority
 1. Parentheses :: ()
 2. Unary minus :: -5
 3. Multiplication, Division, and Modulus
 4. Addition and Subtraction
- For operators of the *same priority*, evaluation is from *left to right* as they appear.

42

In the last lecture, we were introduced to the precedence among the operators, and we had seen that parentheses has got the highest precedence followed by unary minus then multiplication division module, and modulus have got the same priority addition and subtraction are having the next priority. And for operators of the same priority if an expression there are more than one operators which are of the priority then evaluation is done from left to right.

Only one thing we did not mention and that is if we want to change the precedence of the evaluation, then we can always do it using parentheses because we know parentheses is the highest priority one or so, over riding scenario. So, let us look at some examples for this which will make the idea clear.

(Refer Slide Time: 01:30)

Examples: Arithmetic expressions

$a + (b * c) - (d / e)$	$\rightarrow a + (b * c) - (d / e)$	$\text{--- } b * c - x$
$a * -b + d \% e - f$	$\rightarrow a * (-b) + (d \% e) - f$	$d / e - y$
$a - b + c + d$	$\rightarrow (((a - b) + c) + d)$	$a + x - y$
$x * y * z$	$\rightarrow ((x * y) * z)$	$z - y$
$a + b + c * d * e$	$\rightarrow (a + b) + ((c * d) * e)$	

Here are some examples arithmetic expressions; here you can see a plus b times c minus d divided by e. Now according to my precedence rule, this is equivalent a plus as if I have if I have done it hand computation. If I had done by hand computation how would I have done there is a chance of confusion, somebody could have done a plus b and the whole thing multiplied by c etcetera the you know even in school level in order to avoid such confusions we use parentheses. And this is equivalent to this scenario that a; this b c are parenthesized and d divided by e are parenthesized. So, first this will be done then this will be done why b into b times c first, because it is left and this is right, right.

So, first this then this then addition and sub. So, let us see how you will break it down. Again we will have first b times c will be done followed by b times e then we have suppose this result is x and this result is y. Then my expression is a plus x minus y now out of these again these 2 they are the same precedence. So, first this will be done suppose this is z minus y that is how the computation will go on.

Here what is the significance of this? The critical point to note here this part here there is an unary operator. Therefore, during computation what is it means the computer will automatically assume that this parentheses are there alright. If you had not wanted that then you better write in some other way put it some other way. So, that the confusion is not there. The computer will not allow the compiler will generate the code in a way that it will be treated as this a times first minus b will be done, then this one is the highest

precedence next highest precedence. So, this will be done, then multiplication between these 2 will be done now I am sorry I am sorry here normally if I had done this what would have happened.

(Refer Slide Time: 04:07)

Examples: Arithmetic expressions $b=5$

$a + b * c - d / e$	$\rightarrow a + (b * c) - (d / e)$	$\left[\underbrace{a * (-5)}_{=} + \underline{(d \% e)} \right] - f$
$a * -b + d \% e - f$	$\rightarrow a * (-b) + (d \% e) - f$	
$a - b + c + d$	$\rightarrow (((a - b) + c) + d)$	
$x * y * z$	$\rightarrow ((x * y) * z)$	
$a + b + c * d * e$	$\rightarrow (a + b) + ((c * d) * e)$	

First of all I will be doing minus b and by capital letter I am writing some constant value alright. So, b is right now is 5. So, first after computing this it would be a times minus 5 because b was 5 plus suppose d was 100 and e was 9. So, what is my modulus 1 right. So, then I am I will be now at this point, I have got b modulus e minus f.

Now, at this point which one has got the higher priority? This and this, now since now I have put a parentheses here therefore, if I put a parentheses here then this will be done first otherwise this would be would have been done first; because this multiplication has got the same priority as this and put it in a place to write now. So, similarly here a minus b plus c plus d.

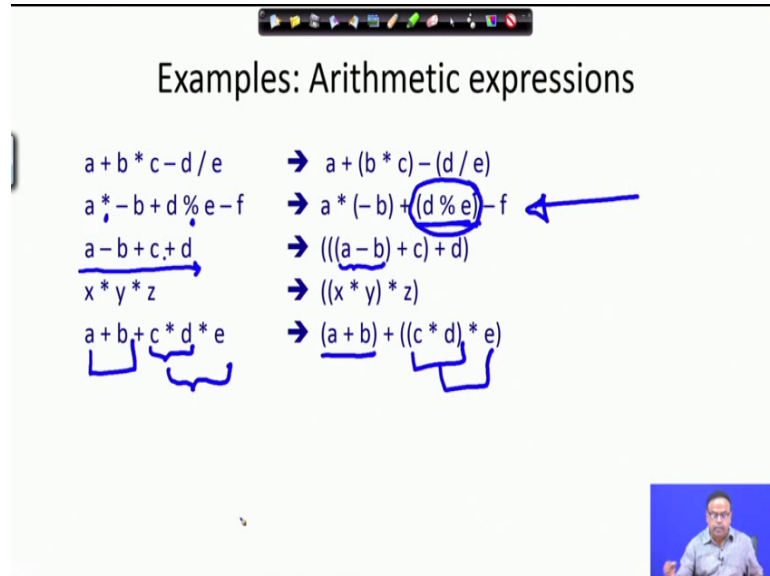
Now, here these are having the same priority left to right, now I want it would to be done in a particular order therefore, I have put the parentheses in such a way that a minus b should be done first, then a minus b plus c will be done first then d will be added to that. That would have been done anyway why because since these are of the same priority that would be done left to right. So, first this would be done, then this would have been done then this would have been done either way.

Here also the same I can write in this way, but that is the precedence is being shown in the form of bracket.

(Refer Slide Time: 06:23)

Examples: Arithmetic expressions

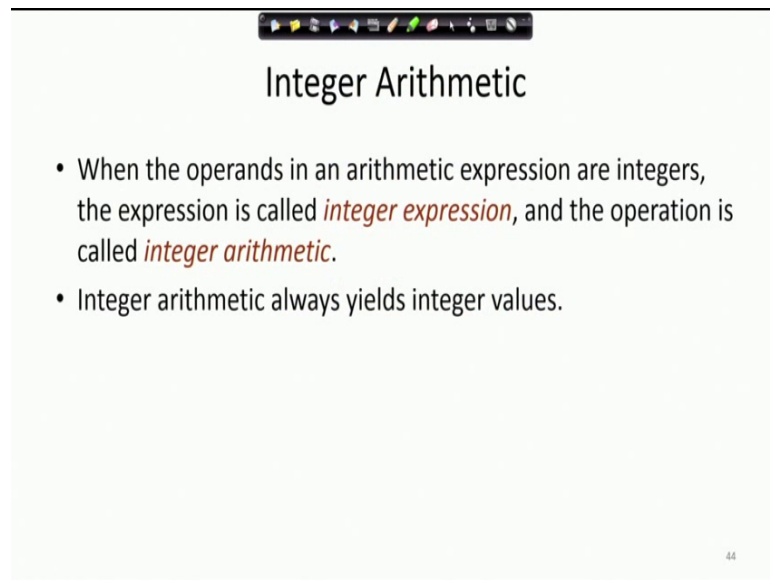
$a + b * c - d / e$	$\rightarrow a + (b * c) - (d / e)$
$a * -b + d \% e - f$	$\rightarrow a * (-b) + (d \% e) - f$ ←
$a - b + c + d$	$\rightarrow ((a - b) + c) + d$
$x * y * z$	$\rightarrow ((x * y) * z)$
$a + b + c * d * e$	$\rightarrow (a + b) + ((c * d) * e)$



But here this is an example, where I use parentheses to over write the precedence. The precedence between this multiplication and this modulus was the same, but just by putting this parentheses there, I have forced this to be computed first. Similarly here now normally given this, this would be done first, then this would have been done, then this would have then then additions would have been done.

But I have over written that by saying that wait first you do it a plus b, then you multiply c and d multiplied e that is there anyway, and then you do the addition, but this is actually over writing the normal precedence. So, you should I mean with a little bit of practice we will initially we will make some mistakes, but then gradually with practice that will go away n and all of you will be able to write it fine.

(Refer Slide Time: 07:32)



Integer Arithmetic

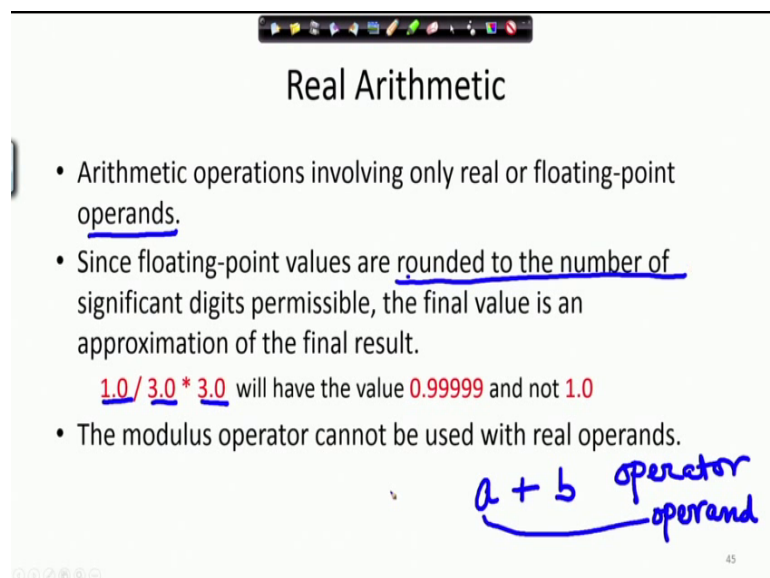
- When the operands in an arithmetic expression are integers, the expression is called *integer expression*, and the operation is called *integer arithmetic*.
- Integer arithmetic always yields integer values.

44

Now, we are coming to a very important concept of this arithmetic expression evaluation integer arithmetic.

What does it mean? It means that when the operands in an arithmetic expression are integers, then the expression is called an integer expression and the operation is called the integer arithmetic. Integer arithmetic always deals integer values, it will always yields I am sorry it will always yield integer values. So, it will be clearer when we take some examples.

(Refer Slide Time: 08:20)



Real Arithmetic

- Arithmetic operations involving only real or floating-point operands.
- Since floating-point values are rounded to the number of significant digits permissible, the final value is an approximation of the final result.
 $1.0 / 3.0 * 3.0$ will have the value 0.99999 and not 1.0
- The modulus operator cannot be used with real operands.

$a + b$ operator operand

45

On the other hand real arithmetic is arithmetic operation involving only real or floating point values.

For example here you can see this expression 1.0 divided by 3.0 multiplied by 3.0. Now all these are operands now here is I would like to introduce a term, when I say a plus b then this plus I call it as an operator, and these I call as operand on which the operation is being done. Now in real arithmetic all the operands are floating point numbers.

Now, sometimes the floating point values are rounded to the number of significant digits because it may be 7.5976 like that you can go on, its often rounded to the number of significant digits permissible in the particular machine, we get an approximation of the results. Now one thing that we have to remember is that the modulus operation is not defined over real operands.

(Refer Slide Time: 09:55)

Real Arithmetic

- Arithmetic operations involving only real or floating-point operands.
- Since floating-point values are rounded to the number of significant digits permissible, the final value is an approximation of the final result.
 $1.0 / 3.0 * 3.0$ will have the value 0.99999 and not 1.0
- The modulus operator cannot be used with real operands.

45

So, I am not allowed to do 3.5 modulus 2.1 that is not allowed. The modulus operation is allowed only for integer arithmetic.

(Refer Slide Time: 10:17)

Mixed-mode Arithmetic


- When one of the operands is integer and the other is real, the expression is called a *mixed-mode* arithmetic expression.
- If either operand is of the real type, then only real arithmetic is performed, and the result is a real number.

$25 / 10 \rightarrow 2$
 $25 / 10.0 \rightarrow 2.5$

$25.0 / 10.0 = 2.5$

Some more issues will be considered later.

$25 \% 10 = 5$ $25 / 10 = 2$



Now, we can also have mixed mode arithmetic, where some of the operands are integers and some are float or real in the case we call it a mixed mode arithmetic expression. If any of the operand is of real type then only real arithmetic is performed and the result is a real number this is this example will make it clear.

Now, you see here look at the first one 25 divided by 10, both the operands this operand and this operand both these operands are integer therefore, when I divide by divide them I get an integer quotient or integer result whereas, if one of the operands was integer and one was real if at least if even one is real, then the result will be real. So, what is happening here? This is becoming like 25.0 divided by 10.0 so that will become 2.5 alright.

Now, here on the other hand since both are integers the result will be integer. So, let me just put in one twist on this. Suppose now here 25 divided by 10 is actually what? Is a if I divide it, it is also 2.5 right 2.5, but since this operator is nothing, but an integer division. So, it will in out the quotient and the quotient is an integer the quotient is 2, if I had done 25 modulus 10 what would my result be? The result would be 5 as a integer right. So, we could understand what is meant by mixed mode real mode, and integer mode of arithmetic we will see some more things later.

(Refer Slide Time: 12:44)

Problem of value assignment

- Assignment operation
variable = **expression_value**;
or
variable1 = **variable2**;
Data type of the **RHS** should be compatible with that of LHS.

$v = u + f * t ;$

$v = u ;$

value

value

Now, there are some problems of value assignment for example, in assignment operation we actually take the expression value and assigned it to a variable or we assign a particular variable to another variable. For example, we could have as we have done say something v is u plus f times t , this is an expression value is going first it will be computed and the value will go to v . So, the value is going there right on the other hand I could have done v assigned u ; that means, the value of u is going to the this variable here also, this value is going here the value of an expression here just a value of a variable. But the most important point is that the data type of the right hand side should be compatible with that of the left hand side ok.

(Refer Slide Time: 13:54)

Problem of value assignment

- Assignment operation
variable = expression_value;
or
variable1 = variable2;
Data type of the RHS should be compatible with that of LHS.

int v;
float u;
v = u;
u = 25.7
v = 25

Now, if for example, I say v assigned u and u is a float and v is an int, then may I may have problem what type of problem can I have? Suppose u is 25.7, I assigned this to v. So, v can only take an integer value. So, in most of the cases we will get an error why? Because you know that depending on the computers, depending on the different architectures integers are often 2 bytes and floats are given 4 bytes.

So, you are trying to track in a large number I mean a 4 byte into a 2 byte space the space is not there. So, the errors can come, in some cases may be they will cut it out and represent it as ignore this part and you will not understand it will just assign 25 to v, that is also an error that is skipping in now this is this error will go unnoticed. If the compiler catches it which is mostly the case, in that case you will you will the compilation error will occur and we will understand that it is not working alright.

(Refer Slide Time: 15:40)

Problem of value assignment

- Assignment operation
variable= expression_value;
or
variable1=variable2;
Data type of the RHS should be compatible with that of LHS.

e.g. four byte floating point number is not allowed to be assigned to a two byte integer variable.

47

For example as I said the 4 bytes floating point number is not allowed to be assigned to a 2 byte integer variable.

(Refer Slide Time: 15:46)

Type Casting

```
int x;  
float r=3.0;  
x= (int)(2*r);
```

(int)

```
double perimeter;  
float pi=3.14;  
int r=3;  
perimeter=2.0* (double) pi * (double) r;
```

The slide includes a diagram showing a 2-byte memory cell for variable 'x' and a 4-byte memory cell for variable 'r'. A blue arrow points from the 'x' variable in the code to its 2-byte memory representation. The 'r' variable is represented by a 4-byte memory cell. The code examples show type casting from float to int and from double to float.

So, sometimes what we do this is an new concept called type casting, look at this example. Here we have defined x to be integer. So, x have got 2 bytes, suppose that I show 2 bytes for x and r is floating number which has been represented through 4 bytes. Now sometimes what we do we type cast this thing 2 times r will be what 2 times r will be float? Because the r is real and. So, if one of the operators are real, then it this part

will be a real arithmetic, but then that result and casting back to int. So, this operation is called the type casting operation; that means, whatever I have I have casting that o fit it in a way. So, that it can fit in as an integer ok.

Here is the another example say perimeter has been defined as double; that means, it can have twice the floating point space. Pi is a float three point one 4 now you see this is pi we have encountered that pi earlier, but here this pi is not the constant which has defined as the pi, but pi is a variable, which has been initialised at the time of declaration and r is an integer.

Therefore when I am computing perimeter is twice pi r. So, what am I actually giving?

(Refer Slide Time: 17:31)

Type Casting

```
int x;  
float r=3.0;  
x= (int)(2*r);
```

$2\pi r$

```
double perimeter;  
float pi=3.14;  
int r=3;  
perimeter=2.0* (double) pi * (double) r;
```

48

I am trying to compute $2\pi r$ right the circumference of a circle. So, twice 2 is real now pi dot r perimeter is doubled. So, I want to have perimeter with a larger accuracy therefore, I first make r to be doubled, I type cast this integer to a double retype, I type cast this float to a double type then I multiply and get the perimeter. So, this is another very powerful operation, which we often need for scientific computation.

(Refer Slide Time: 18:18)

Type Casting

```
int x;  
float r=3.0;  
x= (int)(2*r);
```

Type casting of a floating point expression to an integer variable.

```
double perimeter;  
float pi=3.14;  
int r=3;  
perimeter=2.0* (double) pi * (double) r;
```

Type casting to double

So, type casting of a floating point expression to an integer as is shown here, and the type casting to double has been shown here we have already explained that.

(Refer Slide Time: 18:29)

Relational Operators

- Used to compare two quantities.

<	is less than	
>	is greater than	
<=	is less than or equal to	\leq \leq
>=	is greater than or equal to	\geq \geq

Now, we have till now looked at arithmetic expressions, how to form arithmetic expressions, and arithmetic expressions are formed using arithmetic operators. Besides arithmetic operators there is another very important type of operator called the relational operator. A relational operator is used to compare 2 quantities let us see for example, this symbol say very familiar denotes is less than, this symbol which is also familiar shows is

greater than, this symbol is less than or equal to this, sometimes in our normal course we write it in this way, but in a computer we cannot write it in this way we have to write it less than equal to alright.

Similarly, greater than equal to unlike the greater than equal to that we used to write in this way, we have to write here greater than or equal to now this means greater than or equal to that is well known to us.

(Refer Slide Time: 19:58)

Relational Operators

- Used to compare two quantities.
- < is less than
- > is greater than
- <= is less than or equal to
- >= is greater than or equal to
- == is equal to

Handwritten annotations on the slide include: a box around $x = y$, an arrow pointing from x to y with the text $x \leftarrow y$, and the expression $x == y$.

49

Next here is another operator this requires some discussion is equal to. Now earlier this is very important to know than this is a very source of a very common error and common mistake. Typically when we say in our normal arithmetic x equals y we meant that the value of x and the value of y are the same right, but we have seen that in c language this actually means the variable x is being assigned the value of y right.

So, how do I compare whether these 2 variables are equal or not, for that c provides this technique of using 2 consecutive equal it is find to show x whether x is equal to y alright we will come to this in a moment, there is another operator that is not equal to again.

(Refer Slide Time: 21:11).

Relational Operators

- Used to compare two quantities.

<	is less than	$5 < 7$ / $y < x$
>	is greater than	$x < y$ (circled)
<=	is less than or equal to	$x = 5;$ $y = 7;$ $x + y$ <hr/> \downarrow 12
>=	is greater than or equal to	
==	is equal to	
!=	is not equal to	\neq $!=$

49

Let us compare to what we had in school, we used to write something like this in c we used to write it with this exclamation mark followed by equality, this is means it is not equal to alright.


Now, a very important thing comes into the play whenever I am using this relational operators what do I get? If I write x less than y in an now we know that what are these x and y? These are 2 operands and what is this? This is on one operator. So, when I carry out some operation I will be getting some result. So, if I had done x plus y and suppose x was 5 and y was 7, then x plus y would have given me what 12 would have resulted into 12, 5 plus 7.

Now, if I do this x less than y; that means, 5 less than 7 what will this give me? It will any relational operator gives me only true or false 1 or 0. So, when I say x less than y and x is 5 y is 7 it is true therefore, x less than y in this case will return 1.

(Refer Slide Time: 23:07)

Relational Operators

- Used to compare two quantities.

	<p>< is less than</p> <p>> is greater than</p> <p><= is less than or equal to</p> <p>>= is greater than or equal to</p> <p>== is equal to</p> <p>!= is not equal to</p>	<p><u>y=7; x=5;</u></p> <p>y < x → 0</p> <p>y > x → 1/True</p> <p>y <= x → 0</p> <p>y >= x → 1</p> <p>y == x → 0</p> <p>y != x → 1</p>
---	---	--

49

If I write y less than x, it will return me false because y is 7 and x is 5, then this will be false this will return 0 ok.

Similarly, for greater than y greater than x will return me in this case is for this particular value set, y greater than x will give me true one or it conceptually we can say true that this true cannot be represented as true in a computer it is this in c it is represented as 1. Less than equal to in this case what would we have what would have happened? Y less than equal to x is it true or false less than or equal to in this case is y less than x? No is y equal to x? No therefore, it will be false.

Y greater than or equal to x what would have happened? Y is not greater than is greater x not equal to x therefore, it immediately becomes true y is y equal to x in this case no. So, it will return false y is not equal to x is it true or false it is true therefore, it will be one. So, what we could have observed is we can form expressions using relational operators, but these expressions only return the values true and false ok.

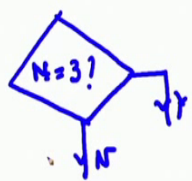
We will see that these relational operators if you recall in our flowchart discussions, we often had some decision box is shown as rhombus and in that decision box we used to say x greater than 10 or say.

(Refer Slide Time: 25:22)

Relational Operators

- Used to compare two quantities.

<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
==	is equal to
!=	is not equal to



49

We used to write something like N equal to 3 right yes or no right we have taken one path for yes one path for no and that can be captured very easily using such relational operators as we will see.

(Refer Slide Time: 25:50)

Examples


$10 > 20$	is false
$25 < 35.5$	is true
$12 > (7 + 5)$	is false

12

- When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared.

$a + b > c - d$ is the same as $(a+b) > (c-d)$

$10 > 17 \rightarrow 0$



Now, here are some other examples or I have already given you enough examples 10 greater than 20 true or false, it is false 25 less than 35.5 yes it is true, 12 greater than 7 plus 5 now here if one of the operands is an expression that will search the evaluated. So,

the arithmetic operators and expressions have higher priority over relational operators alright.

So, here what will happen 7 plus 5 will be first computed. So, that is 12 if 12 greater than 12 no. So, it is false. So, when arithmetic expressions are used, on either side of a relational operator the arithmetic expressions will be evaluated first and the arithmetic expressions will be evaluated in accordance with a precedence that we have already defined. So, if there will be something like this a plus b greater than c minus d that is equivalent to, first I will compute a plus then I will compute c plus d and then suppose a plus b is 10 and c plus d is 17, then this will lead to false first these will be computed then the relational operator will be computed.

(Refer Slide Time: 27:29)

Examples

- Sample code segment in C

```
if (x > y)
    printf ("%d is larger\n", x);
else
    printf ("%d is larger\n", y);
```

Handwritten annotations: $x=10$, $y=15$, 15 is larger

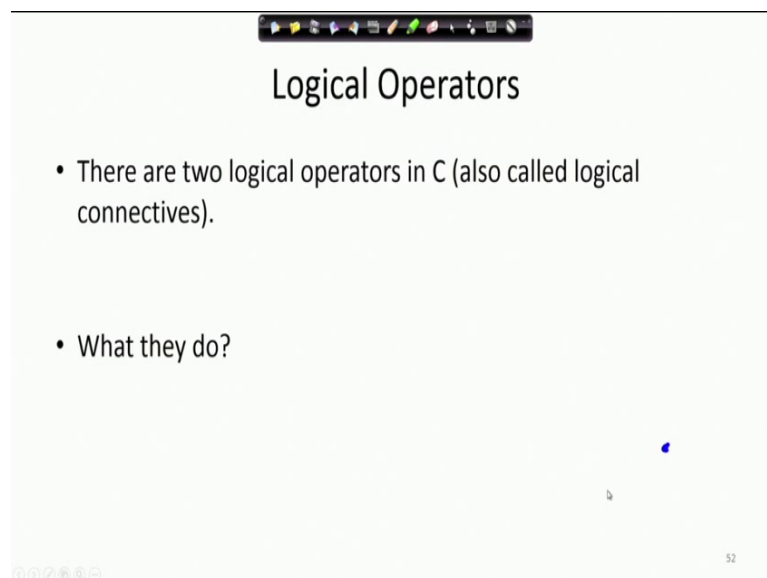
Flowchart: Decision diamond $x > y?$. 'Y' path: print x is larger. 'N' path: print y is larger.

So, here is a quick look at an application of this, suppose I want to implement the flowchart something like this, I have read x and y and I want to do x greater than y if so, yes I will be say I will print x is larger otherwise I will print y is small otherwise I will print y is larger yes. So, that sort of situation how will that translate into c language here you see this part we have not discussed, but is very intuitive you can understand. If x is greater than y, I print what is larger dash is larger then what will come here the value of x. So, I will print say x is 10 and y is 15 n this is x and this is y then what will be printed here? 10 is not greater than 15. So, this part will not be executed will come to here

because x is not greater than y. So, we are taking no path, and we will print dash is larger and in dash what will come the value of y that is 15.

So, what will be printed is 15 is larger. So, that is how we can implement this sort of decision box and when I have that the choice between path based of the decision, that is being a condition is being computed using relational operators which this conditions are telling me which path I will take if the is the situation true if. So, I will take this path otherwise I will take this path this is known as these are (Refer Time: 30:06) of the relational operators.

(Refer Slide Time: 30:13)



So, next class we will start with logical operators, today we have discussed about arithmetic expression how arithmetic expressions are formed using arithmetic operators and what are the very important thing that we discussed today is, you know in the earlier lecture also lecture that what is the precedence till now, why what I have discussed is that what is the precedence of the operators in an arithmetic expression and what is the relational operators. And we have seen that relational operators have the lower priority than arithmetic expressions. So, if I am either side of the relational operators the arithmetic expressions those will be evaluated first, in accordance to the arithmetic operation priority. Next we will come to another very important concept that is logical operators.

Thanks.