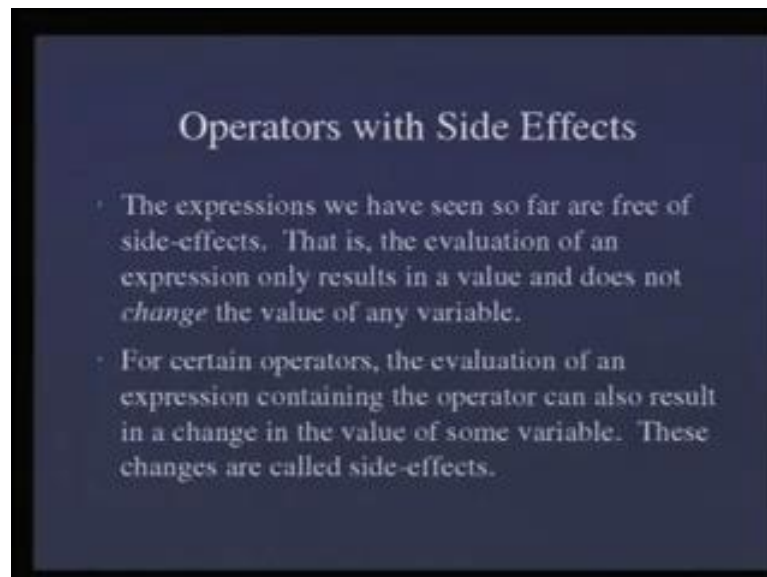


Introduction to Problem Solving and Programming
Prof. Deepak Gupta
Department of Computer Science Engineering
Indian Institute of Technology, Kanpur

Lecture – 8

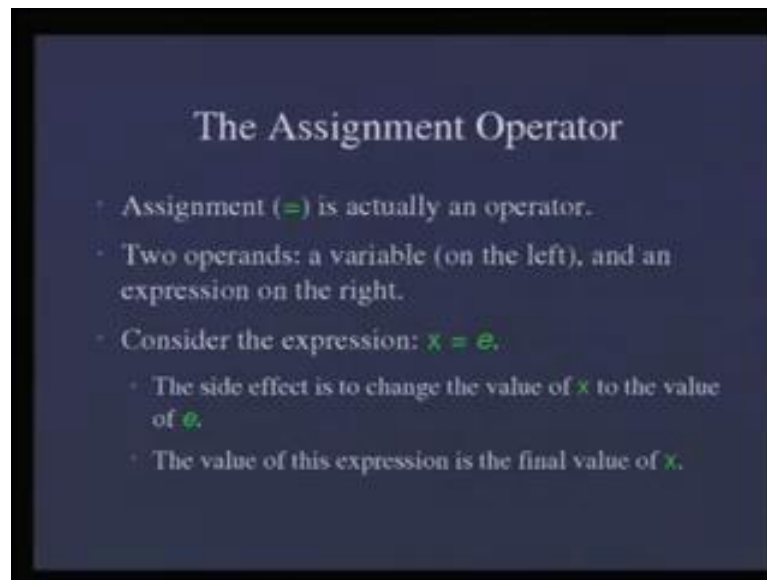
In the last lecture, we had talked about the expressions and operators. We had looked at arithmetic, conditional and logical operators. We will continue the discussion today with operators, which have side effects.

(Refer Slide Time: 00:35)



So far the operators that we have seen when an expression with the operators that are evaluated it results in a value, but as the result of the evaluation of the expression the value of the variable does not change. On the other hand, there are certain operators that we will discuss today where the evaluation of the expression containing such an operator can also result in a change in the value of the some variable. These kinds of changes are called side effects and such operator are called operators with side effects

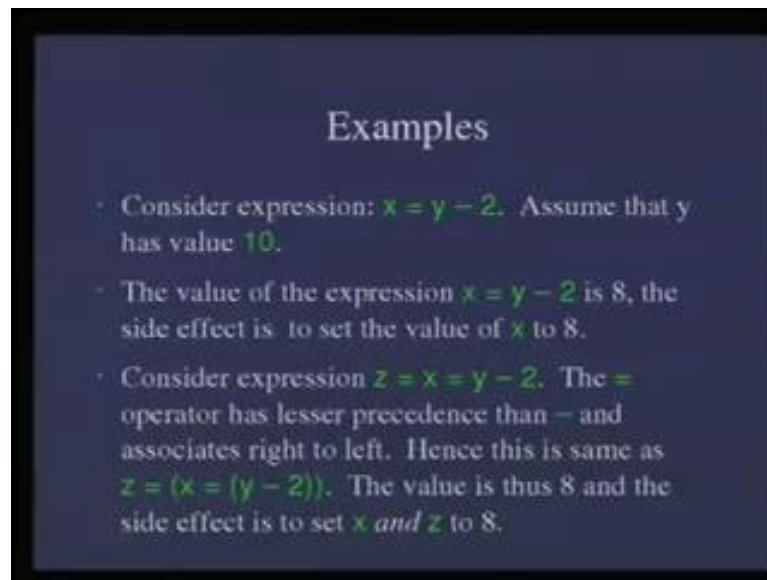
(Refer Slide Time: 01:08)



Let us start with the very familiar operator the assignment operator. We have already used the assignment statement in all the programs that we have written so far. We will be surprised to know that the assignment is actually an operator in C. And it has two operands, the left operand always has to be a variable, let us the variable whose value has to assigned, and the right hand side or the right operand can be any arbitrary expression. So, what we seen to evaluate the assignment operator, we have to defined the effect of evaluating such an expression in terms of what is the value of the expression as well as what is the side effect of the evaluating such an expression.

So, as an example, consider the expression x assigned e , if we evaluate this expression what is going to happen is that the expression e gets evaluated to some value, that value becomes the new value of the x that is the side effect of evaluating the particular expression. And the value that was obtained by evaluating the expression e is also the value of the expression. Thus the value of expression is the same as the final value of the x , which is the value of the expression e .

(Refer Slide Time: 02:27)



Examples

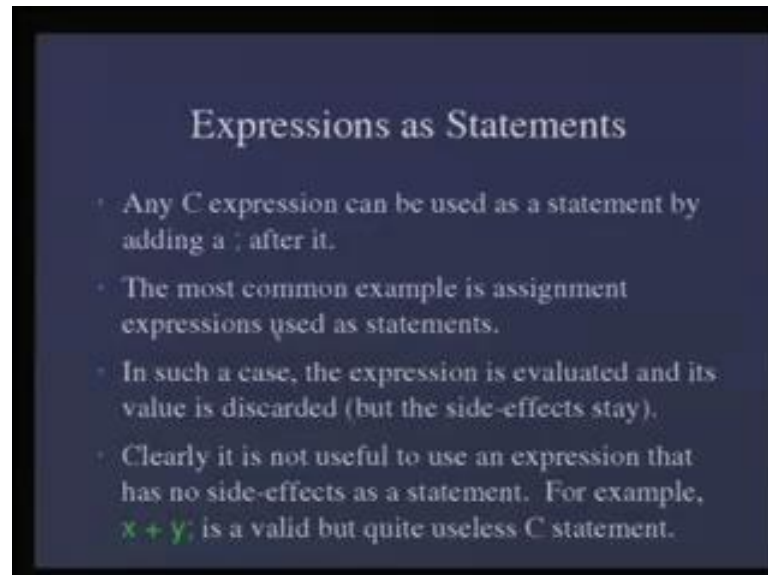
- Consider expression: $x = y - 2$. Assume that y has value 10.
- The value of the expression $x = y - 2$ is 8, the side effect is to set the value of x to 8.
- Consider expression $z = x = y - 2$. The $=$ operator has lesser precedence than $-$ and associates right to left. Hence this is same as $z = (x = (y - 2))$. The value is thus 8 and the side effect is to set x and z to 8.

So, let's see some example. So, consider the expression x assigned y minus two. Let us assume that y has value ten. So, when we evaluate the expression, the expression on the right hand side y minus two gets evaluated which results in the value x . So, the side effects of the special evaluation is that the value of x becomes eight as well as the value of the entire expression x is equals to y minus two is also eight. As it is slightly more complicated example consider this expression z assigned x assigned y minus two, we have to first evaluate the expression of the equal to operator or the assignment operator has the precedence less than that of the minus operator. We will see the complete list of operator precedence and associates slide later on.

The equal to operator has associate right to left not left to right as in the case with the more arithmetic operators. So, which means that the subtraction will be performed first and then the two assignments will happen, and the assignment on the right will happen first, because the associativity of the assignment is right to left not left to right. So, therefore, this expression is same as this bracketed expression. First y minus two is evaluated that assigned to x , and the resulting value of the expression x assigned y minus two is assigned to the vale z . So, when y minus two is evaluated, the value that we get of course is eight again in this case. Assuming that the old value of y was ten and that assigned to the variable x , the value of x becomes eight. Now, the value of this smaller assignment expression is also going to be eight which is the final value of the variable x and that is the value that gets assigned to the variable z . So, that also get the value of

eight and the value of the overall expression also happens to be eight which is the final value of the z.

(Refer Slide Time: 04:41)

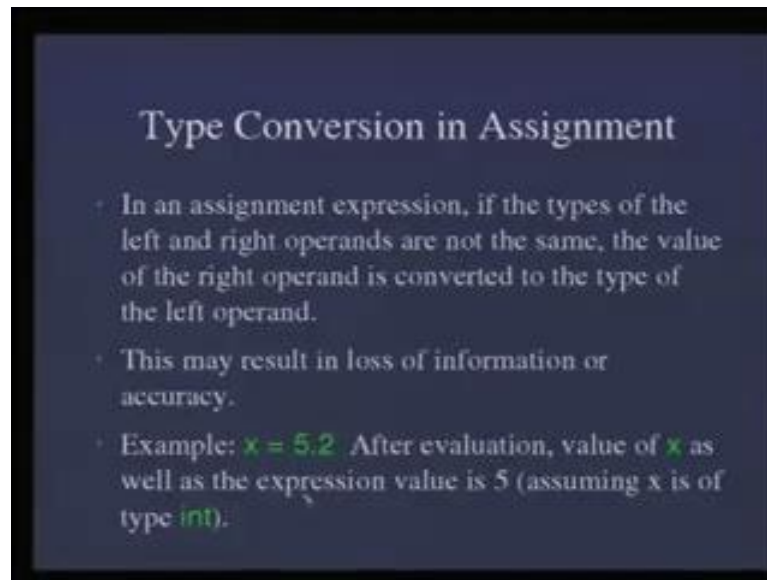


Now, the reason we were able to use assignment a statement is a general rule in C, which is that any C expression can actually be used as the statement by adding a semicolon after it. So, we just saw the expression x assign y minus two if you add a semicolon after this then it becomes a valid C statement. When we use such a statements the meaning is that the expression is evaluated and its value is discarded, there is the value of the expression is discarded. So, this makes sense this is useful only if the expression has certain side effect because it has no side effects only its value is important than it does not makes sense to use that expression as a statement because the value is discarded anyway.

The most common example of course, is the assignment statement which is which is very common in all c program and thus we understand now the assignment statement is actually nothing but an assignment expression followed by a semicolon which converts into this statement. So, either side in such case the expression is evaluated and its value is discarded, but whatever side effect happens as the result of evaluating the expression left it. So, you can see clearly not useful to use an expression that has no side effects as a statement. For example, consider the statement x plus y semicolon. Now, this statement itself is a valid C statement, because x plus y is a valid C expression, but this is quite

useless C statement, because the value of the expression `x plus y` is the target and the expression has no side effects. So, whether you have not quote such a statement in a program that makes absolutely no difference in the working of the program.

(Refer Slide Time: 07:51)



Type Conversion in Assignment

- In an assignment expression, if the types of the left and right operands are not the same, the value of the right operand is converted to the type of the left operand.
- This may result in loss of information or accuracy.
- Example: `x = 5.2` After evaluation, value of `x` as well as the expression value is 5 (assuming `x` is of type `int`).

So, we have now have to talk about type conversion or we move to the implicit type conversion in assignment expression, the rules for type conversion in assignment are slightly different from those that happened in the other arithmetic operator and so on. So, if you recall an operator such as plus minus etcetera, the rule was that as the two operand are of different type then the operand which is of lower type its value gets promoted or gets converted to the higher type.

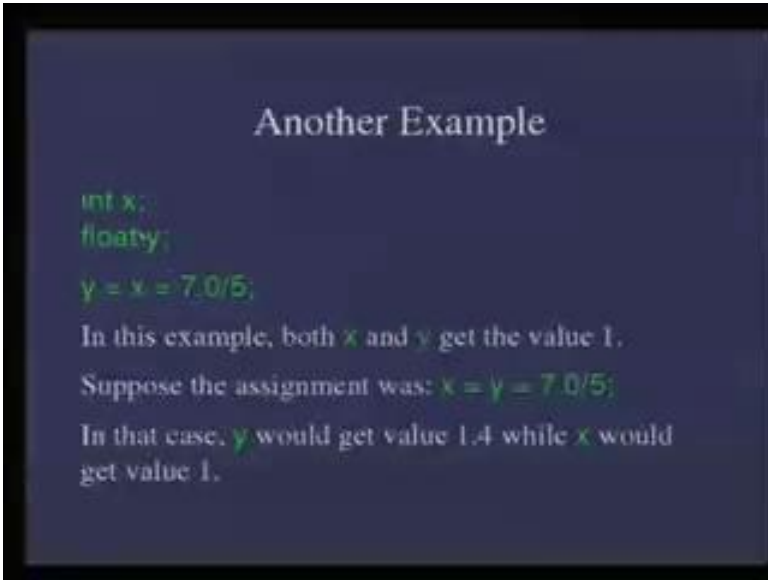
In assignment, on the other hand what happens is that regardless of which of the two types is lower or higher, the value of the right hand operand always get converted to the type of the left hand operand. Because is that the left hand operand is a variable and it is into this variable where the value is going to get stored therefore, whatever the value we get as the result of evaluating right hand side operand must be converted to whatever is the type of the variable. Of course, this may in general result in loss of information or accuracy.

For example, suppose `x` is an integer variable, and we have an assignment `x` assigned five point two you know that five point two is an constant literal of floating point type and is a type therefore, it is double. Now, the double type is higher than the integer type as we

know, but because of the special rule in assignment the type of the expression on the right hand side is going to get converted to the type of the variable on the left hand side, which means that the type of the expression 5.2 which is double is going to get converted to an integer before that the value is stored in the variable x. So, the variable x cannot be of course, stored the value of five point two. So, when five point two double value is converted to an integer, the fractional part is proper and the resultant value is of integer five and therefore, get assign to the variable x.

So, in this case what is happening is the loss of information from five point two the original value what actually get assigned to the integer five. So, while using this kind of assignments where the two operands are of different types one has to aware of this. Similarly, is on the right hand side, we happened to have long integer and on the left hand side we happened to have a variable of type simple int then the possible that the value of right hand side may not actually exist in an integer. If the value happens to be large and cannot greater than the integer then what will happen is that some of the bit will get dropped out and the result will probably not the what we expect. So, one has to be careful in making such an assignment.

(Refer Slide Time: 09:33)



Another Example

```
int x;  
float y;  
y = x = 7.0/5;
```

In this example, both `x` and `y` get the value 1.

Suppose the assignment was: `x = y = 7.0/5;`

In that case, `y` would get value 1.4 while `x` would get value 1.

Here is another example of the same kind. So, in this example x is an integer variable and y is a floating point variable and you have the statement y assigned x assigned seven point zero divided by five and of course, that you know that slash operator has the

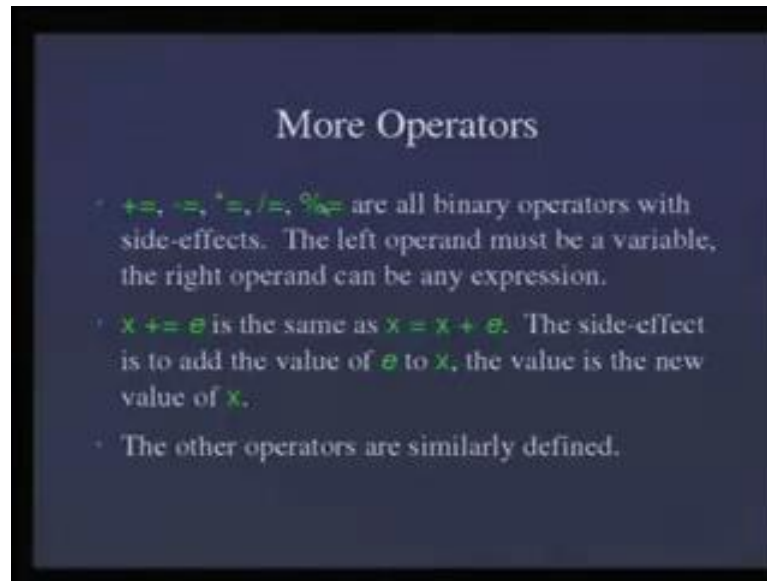
highest precedence. So, the best thing is going to look something like this. The second assignment is going to be evaluated first, because the assignment operator associates from right to left and not left to right. So, the division is carried out first, now in division one of the operand is of size double seven point zero, other is int.

If you remember the rule the value five gets promoted to a double that becomes five point zero, and we divide seven point zero by five point zero that will give you the value one point four. Now, if you look at the assignment which is carried out next, the left hand side x is of type int whereas the expression on the right hand side has the type double and the value one point four. So, what is going to happen in this case, according to the rule for the assignment is that the value on the right hand side is going to get converted to an integer which means that the fractional part will be removed. So, one point four will get converted simply to one. So, the value that x gets is one and the value of the overall expression x is equal to seven point zero divided by five is also one and the size is an integer.

Now, the first assignment y assigned rest of it, it happens. Now the right hand side of this expression has already been evaluated and that has been evaluated to the value one of the i th integer. On the left hand side, we have the variable y , which is of type float. So, the value one of the type integer is going to get converted to float that becomes one point zero. So, y gets the value one point zero. So, in this example, both x and y get the value one. On the other hand, suppose the assignment was x assigned y assigned seven point zero divide by five. So, what is going to happen in this case, this assignment is going to happen first, now seven point zero divide by five is evaluated it is again evaluated to double value one point four.

Now, y is a float the double value one point four gets converted to the float value one point four and that is the value that gets assigned to the variable y . So, y gets the value one point four and the value of this overall expression is one point four of type double of type float and then this assignment happens. Now, where the left hand side is of the type integer and the right hand side is of the type float. So, the right hand side value gets converted to an integer. So, one point four when it gets converted to an integer, the fractional part is dropped off and the resultant value is one. So, the value of x becomes one and of course, that also be value of the overall expression. So, in this case y would get the value one point four while x could get the value one.

(Refer Slide Time: 12:52)



Let us now look at some more operators which are similar to assignment, they all have side effects and. In fact, can be described in terms of the assignment operators. There are five of the plus equal to minus equal to star equal to slash equal to and percent equal to and essentially these combined assignment where some arithmetic operations like plus or minus or multiplication or division are remainder and so on. Of the slide, first four can be used with fourteen points as well as the integer quantity while the sixth one can be used only with the integer quantity.

So, in all these cases, again the left hand operand must be a variable and the right hand operand can be any expression. So, for example, let us look at an expression x plus equal to e this is exactly the same in terms of both value as well as the side effect as the expression x assigned x plus e . Therefore the side effect is to add the value of e to the current value of x , and the new value of x is the value of the expression. And the same with the case with the other operators that we have listed here, the only difference is that instead of this plus, this would be replace with subtraction, multiplication, division or remainder operation as the case it might be.

(Refer Slide Time: 14:10)

Increment and Decrement Operators

The operators `++` and `--` are both unary operators. Both can be used in prefix as well as postfix form but the meaning is different. The operand must be a variable.

Assume `x` is variable of type `int` with initial value x_0 .

Expression	Value	Side-Effect
<code>x++</code>	x_0	$x \leftarrow x_0 + 1$
<code>++x</code>	$x_0 + 1$	$x \leftarrow x_0 + 1$

Another two commonly used operators in C are the increment and decrement operator. These are denoted by plus plus and the minus minus symbol, there should not be any space between the two pluses or the two minuses. Both are unary operators that is there is only one operand, and the operand must be a variable. And the interesting this is that both can be used as the prefix as well as the postfix form, but the meaning in the two cases is different. Prefix form mean that the operator comes before the operand and the postfix means that the operand comes before the operator.

So, let us see what these operators really mean. Let us assume that `x` is a variable of type `int` with some initial value `x zero`. So, to look at the expression `x plus plus` the value of the expression is `x zero` and the side effect is to increment the value of `x` by one that is the new value of `x` is the old value of `x zero` plus one. So, this is the postfix form of the plus plus operator in the prefix form the value of expression is `x zero plus one` and the side effect is the same which is to add one to the whole value of `x`. So, the new value of `x` become the old value that is `x zero plus one`.

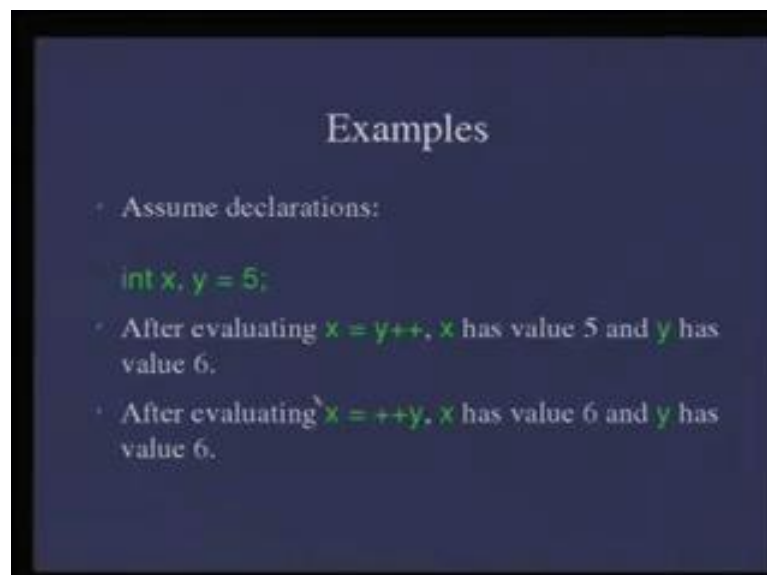
So, the only difference really in these two is the value of the overall expression. One way to understand this is that in the postfix form, we could say the value of `x` is used as the value of the expression and then the side effect happens that is the value of `x` then gets incremented once the value of the expression can be computed. Whereas in the prefix

case here the value of x first gets incremented that is the side effect first happens and then the new value of the x is used as the value of the expression.

So, to clarify these operators, let us take some examples. You can see that the minus minus operator is exactly the same as the plus plus operator except that these pluses are replaced with the minuses that is the side effect is to decrement the value of x by one and the value of the expression is either the old value of x or the new value of x which will be the old value of x minus one. So, here are some examples to clarify this let's assume that x and y are integer variables and the initial value of y is five.

So, let us assume that the expression x is equal to y plus plus is evaluated. So, now, if you look at this expression y plus plus the value of the expression is same as the old value of y right. Because this is the postfix form of the plus plus operator. So, the value is same as the old value the value of the expression y plus plus is the old value of y which is five and the side effect is to increment the value of y by one. So, now what gets assigned into the x is the value of the expression y plus plus which is the old value of y which is five and. So, x gets the value five while y gets the value six after the entire expression has been evaluated

(Refer Slide Time: 17:44)



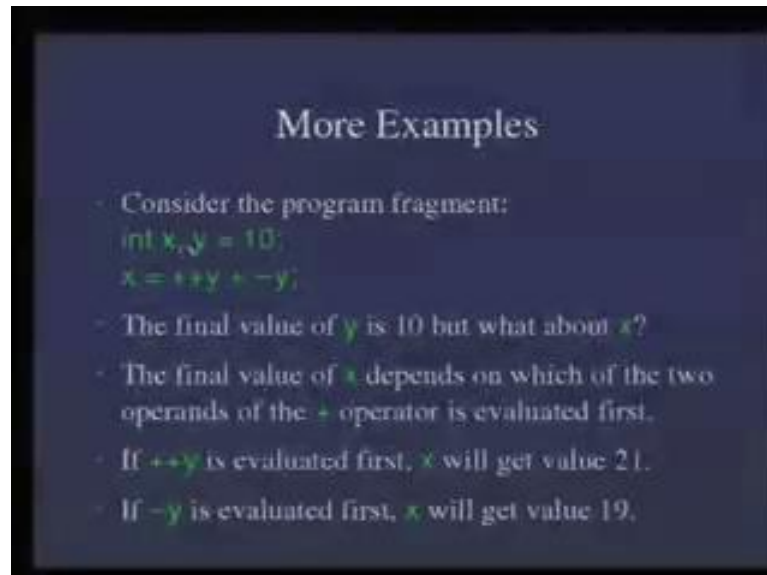
The slide is titled "Examples" and contains the following text:

- Assume declarations:
`int x, y = 5;`
- After evaluating `x = y++`, x has value 5 and y has value 6.
- After evaluating `x = ++y`, x has value 6 and y has value 6.

Now, on the other hand, if we had evaluated the same declaration that the expression x is equal to plus plus y. Now what is going to happen is that plus plus y is evaluated, the value of y is increased by one it becomes six, and the value of the expression is the new

value of y which is also the same. So, therefore, the value that x gets in this case six and not five.

(Refer Slide Time: 18:12)



More Examples

- Consider the program fragment:

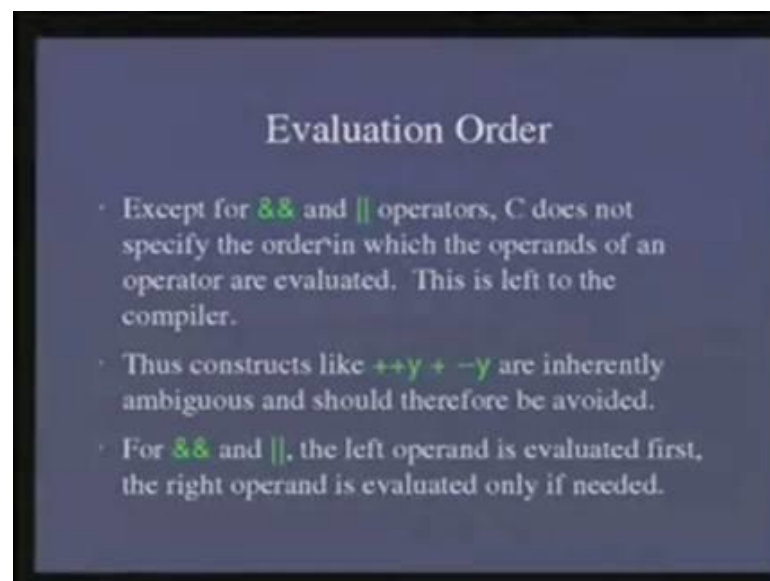
```
int x, y = 10;  
x = ++y + -y;
```
- The final value of y is 10 but what about x?
- The final value of x depends on which of the two operands of the + operator is evaluated first.
- If ++y is evaluated first, x will get value 21.
- If -y is evaluated first, x will get value 19.

Let us consider some more trickier example, look at this program segment. X and y are declared to be integer variable, and y has the initial value ten. And we have the assignment statement x assigned plus plus y plus minus minus y. The question is after the evaluation of the statement or after the execution of the statement what are going to the values of variables x and y. Now with respect to y, you can see that there is one increment and one decrement happening to the value of y, which means the value of y is going to be ten, it is going to be remain ten. Because ten plus one becomes eleven and eleven minus one again become ten. Regardless of the order, in which this two expression two sub expression plus plus z and minus minus y get evaluated, the final value of y will be ten, but what about the value of x the value of x is stands out will vary depending on which of these two expression is evaluated first.

Note that plus plus and minus minus both have high precedence higher than that of the binary plus operator and higher also than the precedence of the assignment operator. So, therefore, these are going to get evaluated before the plus operator, but if you recall the c language does not satisfy or a plus operator which of the two operand is evaluated first. And as turns out in the example, the answer for the final value of the x will be different depending on which of the two is the case.

So, let us assume that plus plus y happens first. So, a plus plus y is evaluated first then what happen, after the evaluation of this expression, the value of y becomes eleven and the value of this entire expression plus plus y, this also eleven, because remember this is pre increment that is the increment happens before the value of the expression is computed. So, the value of this expression is eleven, and y has also changed to eleven and then the expression minus minus y gets computed which means that y gets incremented from eleven to ten. And the value of the expression minus minus y also is the final value of y which is ten which means that x gets assigned eleven plus ten, this expression get evaluated to the eleven and this expression is gets evaluated to ten. So, the result is twenty one. On the other hand, suppose minus minus y was evaluated first. So, indexes it will result in the value nine and then plus plus y is going to get evaluated to the result ten, the final value of x becomes ten plus nine that is nineteen.

(Refer Slide Time: 21:12)



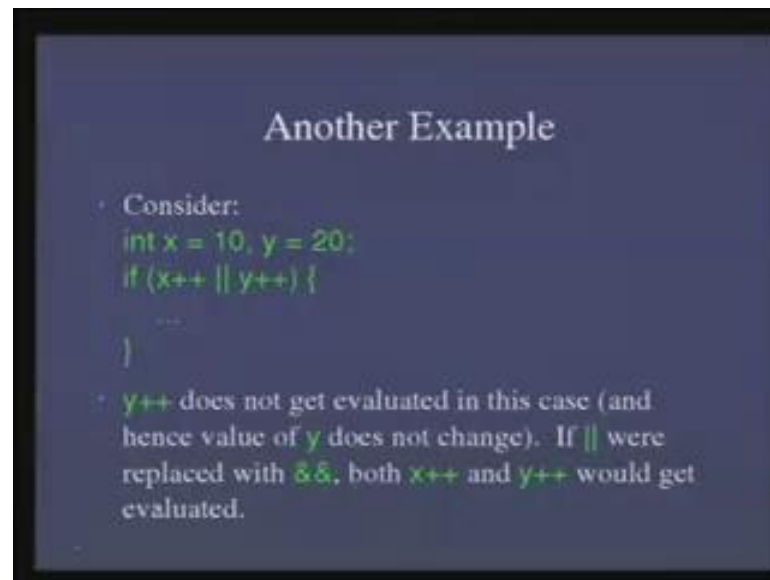
So, in this case, you can see the final value of x will depend upon those or this expression get evaluated first and the c language does not specified as the which one will happen. The step for the two operators these two exceptions we have noted in the last lecture will come back shortly. Except for these two operators, C does not specify the order which the operand of the particular operator are evaluated. This is left to the compiler, which means that some compiler may evaluate the left hand operand first and some compilers may evaluate the right hand operand first. And if you are using an expression, in which the final results can be vary depending on this order depending on

the order in which the two operands are evaluated then what that means is that depending on the compiler that it being used the result might be different.

So, construct like the one like that we saw just now plus plus y plus minus minus y, they are inherently ambiguous they can be interpretation in two different way which are not the same. And therefore, we must avoid them, because our influence is only one of these two possible interpretation, but on different machine on different compiler the same program may actually give a different result. So, therefore these expression, and these concepts should be strictly avoided. For the logical and and logical or operator the c language does define the order in which the two operand are evaluated as the matter of fact be left operand is always evaluated first and the right operand is evaluated only if the result of the expression cannot be computed based only on the value of the left operand.

So, if you remember that for logical and operator is the first operand, the left operand evaluate to then the entire expression must evaluate to false, because for it to evaluate to both the operands must be true. So, the first operand evaluates to false then the second operand will not be evaluated at all, and if the second operand has happen to have some side effects then those side effect will not takes place, simply because the second operand does not get evaluated at all. And similarly for the logical or operator, if the first operand evaluates to true then regardless of the value of the second operand, the entire expression will get the value true, and therefore, the second operand will not get evaluated at all and again if it has side effects then those side effects will not takes place. Therefore, again these operators has to be careful as what the final value of the various variable are going to be.

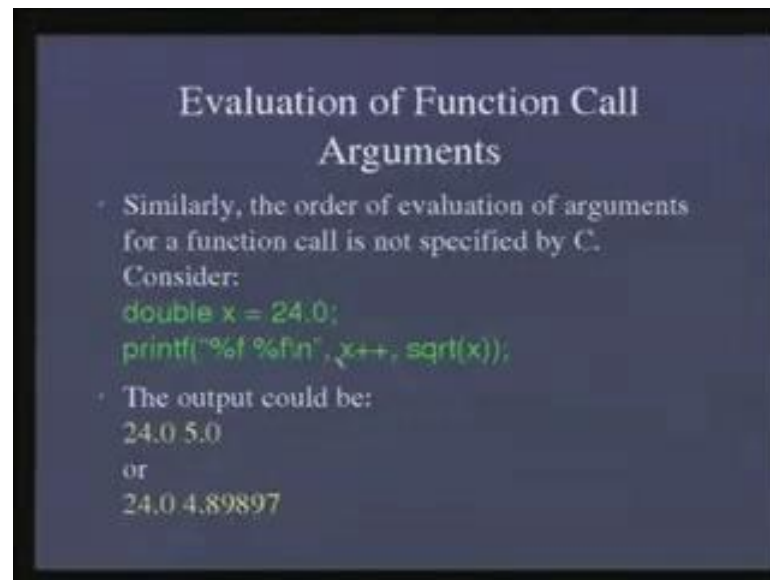
(Refer Slide Time: 23:39)



Let us take the simple example, here x is initialize to ten and y is initialize to twenty and we have an if condition, if x plus plus or y plus plus. So, this is the logical or operator what going to happen is that first operand, which is x plus plus is going to evaluate first. Now the value of x plus plus will be ten, because this is post increment and of course, the side effect is the value of x becomes the eleven. Since the value of this expression turns out to be ten, which is non zero and therefore, considered as true. So, therefore, second operand need not be evaluated at all that is the expression y plus plus will not get evaluated at all. So, and the entire expression will result in the value true, because this first operand is itself true. So, which means that at the end of the evaluation of this entire expression, the result will be true, the value of x will become eleven and the value of y will remain twenty.

On the other hand is inside of the or logical or operator, we had the logical and operator then what should have happened is that x plus plus is evaluated that results in value ten. The x is incremented to eleven, but since this is true the second operand must also be evaluated, because the second operand turns out to be false and the overall result to be false and the second operand turns out to be true the overall result will be true y plus plus is also get evaluated in that case. So, y plus plus id of course, result in the value of twenty and y would get incremented to twenty-one. So, since both are truth over the result would be true, but both x and y would be incremented in that case.

(Refer Slide Time: 25:42)



The slide has a dark blue background with white text. The title 'Evaluation of Function Call Arguments' is centered at the top. Below the title, there is a bulleted list. The first bullet point states that the order of evaluation of arguments for a function call is not specified by C. The second bullet point says 'Consider:' followed by two lines of C code: 'double x = 24.0;' and 'printf("%f %f\n", x++, sqrt(x));'. The third bullet point says 'The output could be:' followed by two possible outputs: '24.0 5.0' and '24.0 4.89897'.

Evaluation of Function Call Arguments

- Similarly, the order of evaluation of arguments for a function call is not specified by C.
- Consider:

```
double x = 24.0;  
printf("%f %f\n", x++, sqrt(x));
```
- The output could be:
24.0 5.0
or
24.0 4.89897

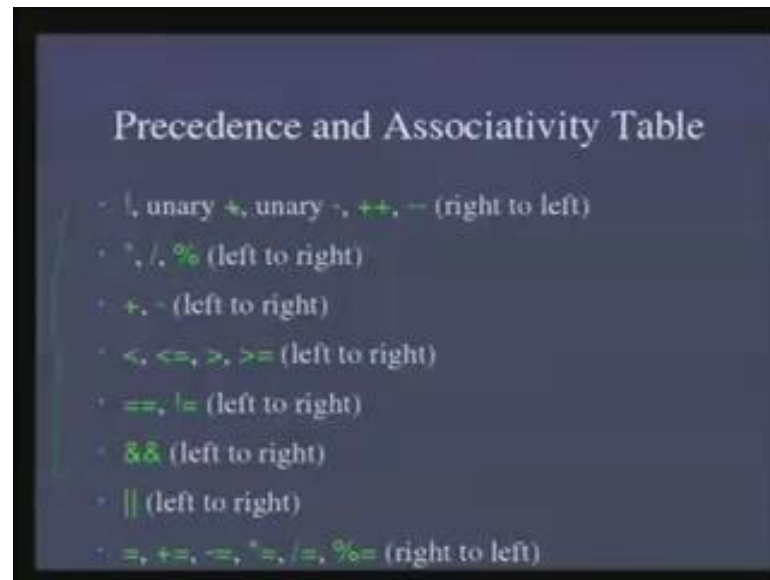
Similarly, the order of the evaluation of the argument for a function call is not specified by C, you know that we used the several library functions and most of our program. And when we pass parameters for arguments to be functioned the order in which these arguments are evaluated is again not specified by the C language it is up to the compiler. So, let us see an example. So, here x has the x is the variable of the type double with the value twenty four, and the printf statement print two double number, the percent f is used t printf an double quantity. The first quantity is x the value of the expression x plus plus and the second quantity is the value of the expression square root x. Now just I said the C language does not specify in what order these two operands are going to be evaluated.

Now suppose the argument x plus plus is evaluated first which means that x gets incremented to twenty five and the value of these expression is twenty four. So, and then afterwards the square root x is computed. Now here the square root of twenty-five will be computed because x is already become twenty-five. So, which will of course, result n the value five and therefore, the output will be twenty four point zero for this parameter followed by five point zero for this value.

On the other hand, the square root x has evaluated first then that will be the square root of twenty four which will be something like four point eight nine seven. And then suppose the second parameter got to be evaluated that could again result in the value twenty, because the evaluation of the first expression did not have any side effect x

remains twenty four. So, when these gets evaluated again the value is twenty four and the side effect is to increment x to make it twenty-five. So, the output would be twenty four point zero followed by four point eight nine eight nine seven and not five point zero.

(Refer Slide Time: 27:56)

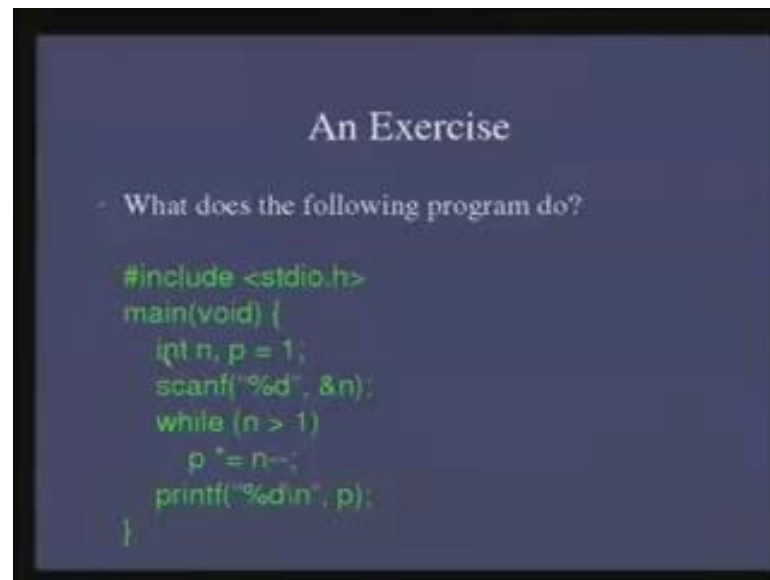


The slide displays a table titled "Precedence and Associativity Table" with a list of operators and their evaluation order. The operators are grouped into nine categories, each with a specific associativity direction. The categories are listed from highest to lowest precedence.

Operator	Associativity
!, unary +, unary -, ++, --	right to left
*, /, %	left to right
+, -	left to right
<, <=, >, >=	left to right
==, !=	left to right
&&	left to right
	left to right
=, +=, -=, *=, /=, %=	right to left

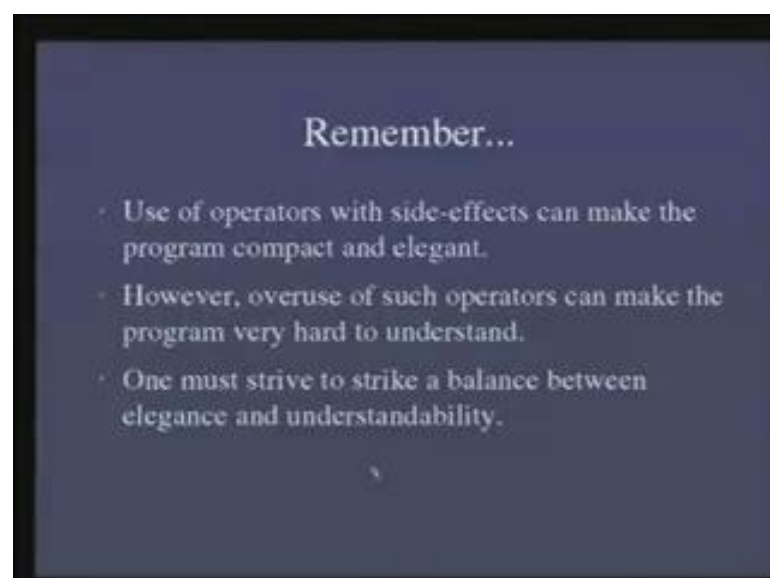
So, we have introduced the many new operators in this lecture. So, where is the somebody of the precedence and associativity rule, as usual the precedence is decreasing from top to bottom. The highest precedence operator are the unary operator we had seen logical not the unary plus and the unary minus in the last lecture, and the increment and the decrement operator are the two new unary operators that we have seen in this lecture. So, all these unary operators have the same precedence, and they associated from right to left. Then come the arithmetic operations multiplications division remainder associativity is left to right, then the addition, subtraction again the left to right, then less than less than equal to greater than, greater than equal to left to right again. Then equals and not equals left to right, the logical AND and the logical OR both as left to right, and finally, the assignment and the assignment like operators like equal assignment plus equal minus equal star equals and so on.

(Refer Slide Time: 28:59)



Here is little exercise at end of this lecture. Please look at this program, make a note of it and try to find out what it does. So, you see that the first part is clearly split of we declare two variable n and p, p is initialize to one, the value of n is read from the terminal. While n is greater than one, the loop is executed; and the loop bodies p star is equal to n minus one semicolon. Note that the star equals to is an operator and this holds the entire thing p is an expression and therefore, we are putting the semicolon after it we can use it as the simple C statement. So, try to figure out what this program is doing, you will find that it does something very familiar.

(Refer Slide Time: 29:49)



So, at the as a caution at the end of the lecture, you have to introduce to many interesting operators for side effect, and as you start using these operators, we will find use of these operators can make the program very compact and elegant in many cases. However, overuse of operations is not advisable, because if you use these operators too often, you tend to make the expression very very complicated and hard to understand that will mean that the program itself can become very hard to understand and therefore, of course, try to balance between elegance and understandability.