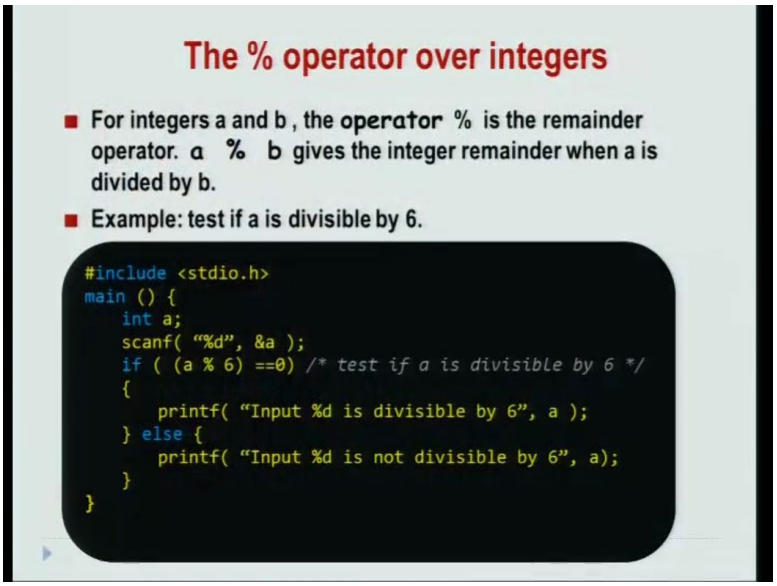
Introduction to Programming in C
Department of Computer Science and Engineering

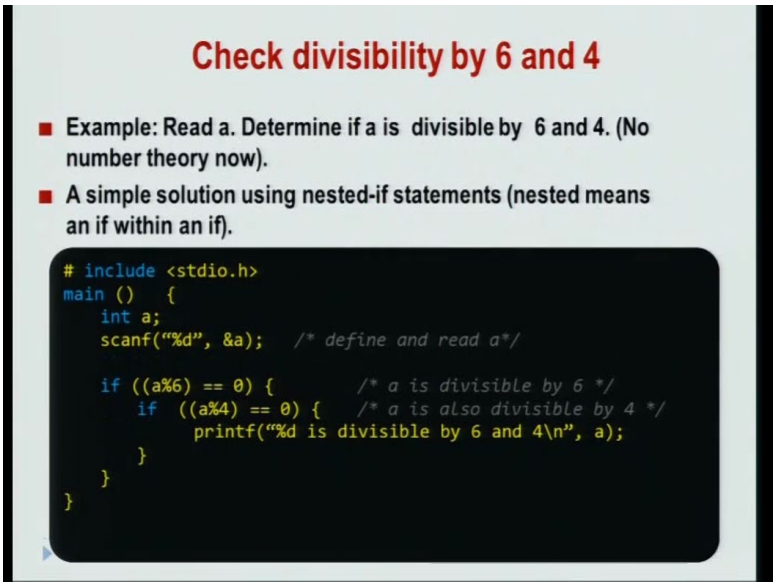
We have seen comparison operators, like less then, equal to, less than or equal. to and so, on. We will see bunch of few more operators in this session. So, let us consider the modulo operator which we have already seen in when we discussed utility in GCD.

(Refer Slide Time: 00:27)



So, a % b gives the reminder when a is divided by b. So, suppose we have the following problem, we get a number a and we want to check whether the given number is divisible by 6. If it is divisible by 6 a % 6 will be 0 the reminder will be 00. So, we will write a simple code, you have int a, a is of type int. Then, scan the number using scanf("%d", &a). And then, you test whether a is divisible by 6 to test whether a is divisible by 6 you check whether a mod 6 is 0. If it is divisible, you say that input is divisible by 6 %d a. Otherwise, else, printf the input is not divisible by 6 very simple operation.

(Refer Slide Time: 01:29)

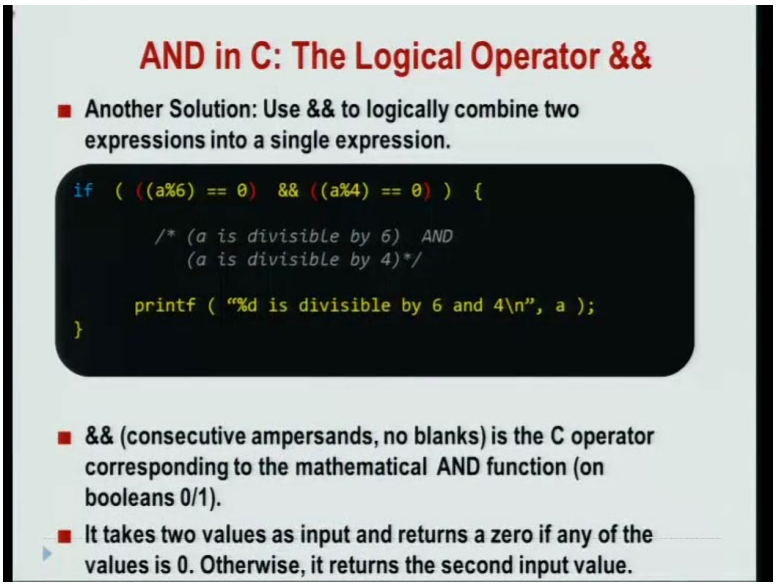


Now, let us make it slightly more elaborate. Suppose, you have to test whether, this a slight variant. Suppose, you have to test whether a give number is divisible by 6 and by 4 two numbers. How do you do this? So, you scanf the number and you test whether a is divisible by 6. So, a % 6 is 0. If that is true, then you also check whether a % 4 is 0. If both are true, then you print that the given number is divisible by 6 and 4. So, percentage is divisible by 6 and 4 a.

So, you can argue about this program and see that, if it is divisible by 6, but not by 4 then, it will enter the first if, but not enter the second if. Therefore, it will not print that it is divisible by 6 and 4. Similarly, if it is not even divisible by 6 it will not even enter the first if condition. So, you will in any case not print that it is divisible. So, convince yourself that this particular code will print a number is divisible by 6 and 4 if and only if the given number is divisible by both 6 and 4.

Now, that piece of code was slightly long is there any way to write the same code with a fewer number of lines. And for this c provides what are known as logical operators. Now, there are three logical operators in Boolean logic which are Boolean AND, Boolean OR and Boolean NOT. So, there are three logical operations AND, OR and NOT and C provides all of them. So, the same if condition that we wrote before, we could have easily said if it is divisible by 6 and if it is divisible by 4 then print the output.

(Refer Slide Time: 03:44)

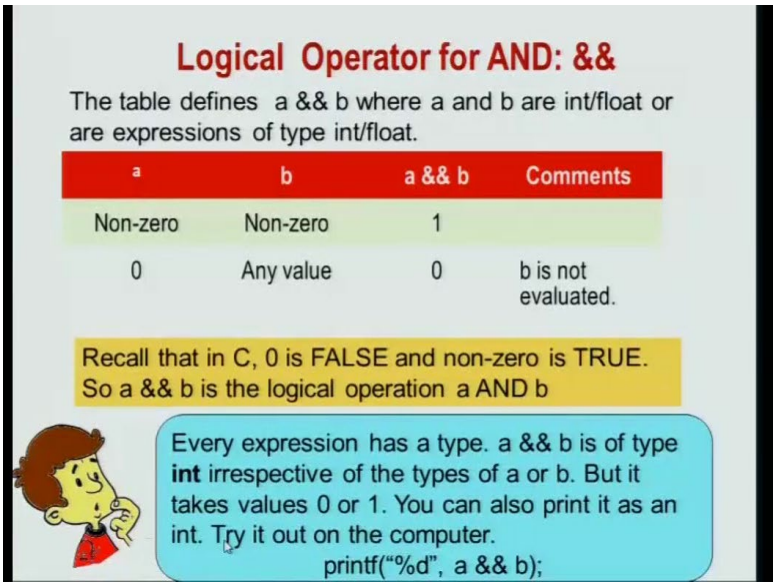


So, for this C provides an operator which is the Boolean AND operation. So, the Boolean AND operation in C is given by two ANDs. So, by know you should be familiar with the fact it certain operations in C have repeated characters. For example, we already have seen the equality operations which was equal, equal. Similarly, the Boolean operation and it is actually the and symbol on the keyboard. But, you have to have two of them that represents the logical AND.

So, this expression says if a % 6 is 0. So, this expression is what test for a is a multiple of 6. And this is the expression which test whether a is the multiple of 4. So, if both conditions are true, then you say that the given number is divisible by 6 and by 4. So, consecutive ampersand signs, that is the and symbols without any blanks in between is the C operator corresponding to the mathematical and the logical AND function.

So, it takes two values as input and returns a 0. If any of the values is 0, if both values are 1 then it returns a 1. So, this is the same as the logical AND. If either of them is 0 then the result is 0, if both of them are 1, then the result is 1.

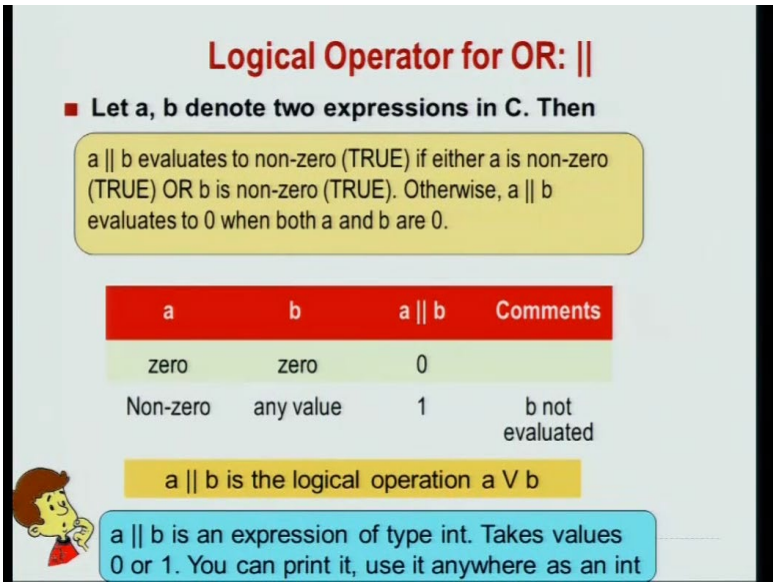
(Refer Slide Time: 05:39)



So, the truth table for the operation AND is as follows if a is a non-zero value and b is a non-zero value, then C considers that both are true. So, the output value is of a and b is 1. If a is 0 and b is any value at all the output is 0 and b is not evaluated. So, this the same as logical end. The only think to notice that, if in evaluating a and b you already know that a is 0, then you know the result is 0. So, C will not bother to evaluate b. Because, it knows that the result is already 0.

Every expression has a type a and b is of type int regardless of the types of a and b. This is because a and b is a logical assertion. The type of a logical assertion is that, it is either true or false, it that it corresponds to a Boolean value. Therefore, at the type of an a and then b regardless of what a and b are the result is always 0 or 1. So, it is of type int. Now, you can print the result as an int, you can say printf %d a and then b.

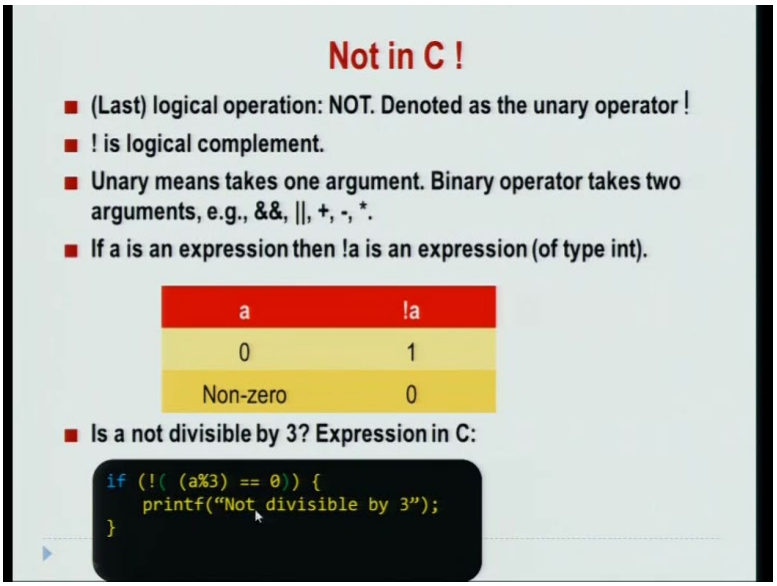
(Refer Slide Time: 07:04)



Now, there are three logical operations as I mention. So, there is also OR in c it is denoted by two vertical bars which are there on your keyboard. So, a or b which is a || b evaluates to non-zero if either a is non-zero or b is non-zero. If both of them are zero, then the result is zero. So, this the meaning of a logical OR operation, if both of them are false, then a or b is false. If at least one of them is true then a or b is true.

So, you can write the truth table for that. If a and b are 0 then the output is 0, if a is non zero and b is any value. Then, in already know that the output of a or b is 1. So, the output is 1 and b will not be evaluated. This is similar to in the case of AND. If a was 0 and b was any other value, then you know that the output of and is 0. Therefore, b will not be evaluated. Similarly, if here if a is non-zero, then you know the value is 1. So, b will not evaluated and as before a or b is of type int.

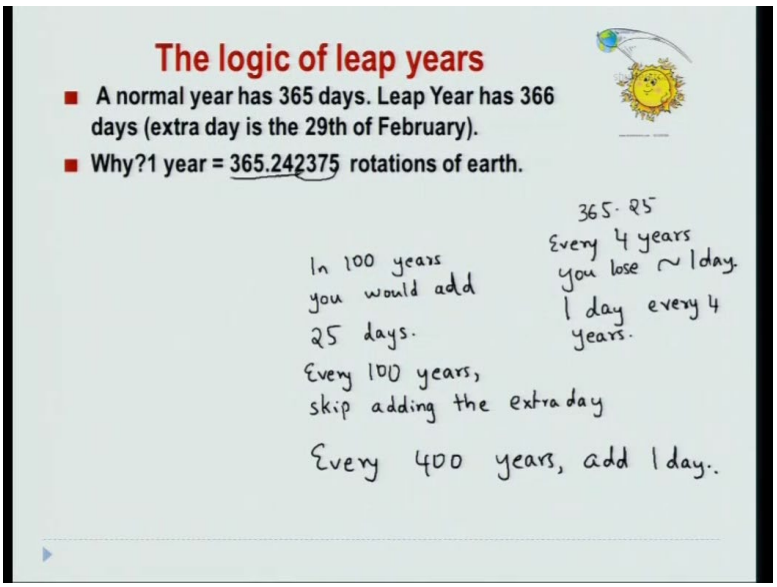
(Refer Slide Time: 08:33)



So, the third logical operation is NOT. Now, NOT in C is denoted as the exclamation mark. So, let us see an example of that. So, NOT is the logical complement and it takes only one argument, this is different from the previous two that we have seen a or b and a and b both took two arguments it is not takes only one arguments. So, it is called a unary operators. So, NOT of a is an expression of type integer and the value is the negation of a. So, if a is 0 NOT of a will be 1 and if a is non zero then NOT of a will be 0.

So, for example, if I want to say that a is not divisible by 3 I will just write NOT of a mod 3 equal to 0. You know that a mod 3 equal to 0 test for a being divisible by 3. So, negation of that it will say that the given number is not divisible by 3.

(Refer Slide Time: 09:43)



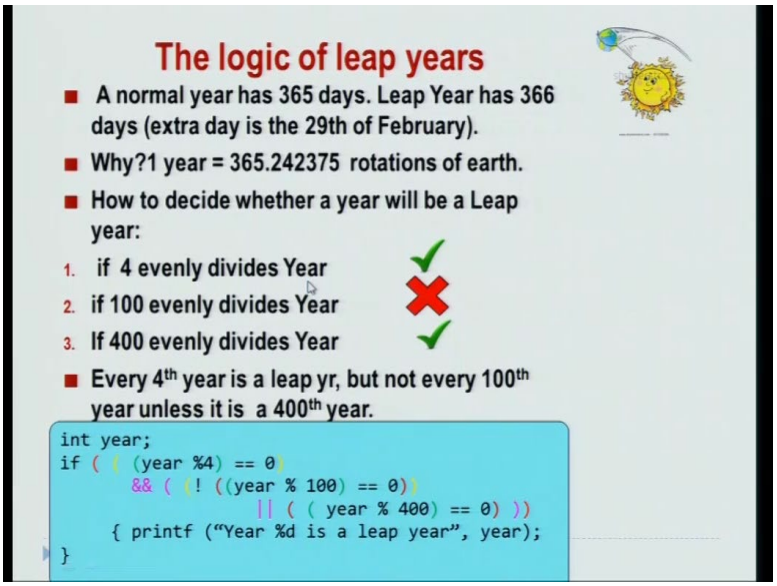
Let us finish this by slightly complicated example which is that of leap years. So, I am given a particular year number and I want say whether the given number is a corresponds to leap year or not. Now, what is a leap year it is that you add a few years will have February 29th in February all other years will have 28 days in February. So, what is the logic of a leap year. So, roughly an average solar year is 365.242375 rotations. So, in particular is not an integer.

So, we normally say that year has 365 days that is not quite true, this it is a rounding. So, how much are we losing. So, you can calculate it as follows. The remaining number after the decimal point is a roughly 0.25. So, at a rough cut let say that every 4 years. Because of this 0.25 you will lose one day. So, every year you are losing about a quarter of a day. So, if every 4 years you will add a day. Now, when you do that you go back to... So, just a minute.

So, you have 365 point let us say 25. So, every 4 years you would lose about a day approximately 1 day. So, in order to compensate for that you add 1 day every 4 years. So, in 100 years you would have added 25 days. But, that is 1 day too much. Because, remember that this number is only 365.24 something. So, in 100 days you should have added only 24 days. But, now you added a 25 days. So, to compensate for that every 100 year skip adding the extra day.

So, every 4 years you have add 1 extra day,, but every 100 years every 100th year you do not add that extra day, you skip it. Because, you would have added 1 more day then you. And then again you can look at what remains, what remains is roughly .24 which means that every 400 years if you do this adjustment you are losing about a day. Because, every 100 years you are losing about quarter of a day from this 0.2375 part. So, you do the same logic again every 400 years. So, every 400 years add an extra day. So, this is the logic of the leap year that we all know. So, how do you decide whether year will be leap year.

(Refer Slide Time: 14:01)



So, the logic that I have outlined just now says that, if a year is a multiple of 4 then it is a leap year. But, if a year is a multiple of 100 as well then it is not a leap year. But, if it is a multiple 400 than it is a leap year. So, here is a pretty complicated expression. So, every 4th year is a leap year. But, skip every 100th year unless it is also a 400th year. And you can write this expression in C, it is slightly complex has you can imagine.

So, if the first line the first expression says that, wise year is a multiple of 4. So, if year is divisible by 4 also the following should be true, it should not be a multiple of 100 unless it is a multiple of 400. So, it should not be divisible by 100 that should be true or it should be true that, it should be a multiple of 400. For example, if you have 400 then it is a leap year. So, what will happen is that year modulo 4 400 modulo 4 is 0.

Then, what happens is that you have 400 modulo 100 that is 0. So, this is equal to 0 that is 1 NOT of 1 is 0. So, this part is entirely 0, but it is divisible by 400, 400 divided by 400 is 0. Therefore, this part is true, this or 0 or 1 is true. Therefore, the whole expression becomes 1 and 1. So, it is true. So, this logical expression slightly complicated logical expression encodes the logic for saying that the given year is a leap year. So, try this out yourself this is a slightly tricky expression. And convince yourself that, this exactly encodes the logic of the leap year.