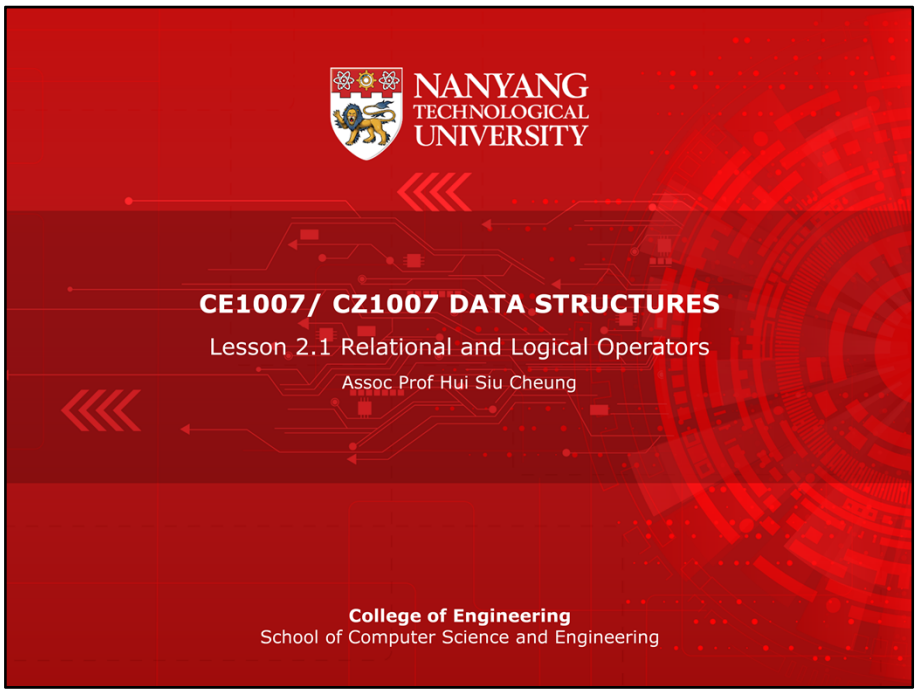
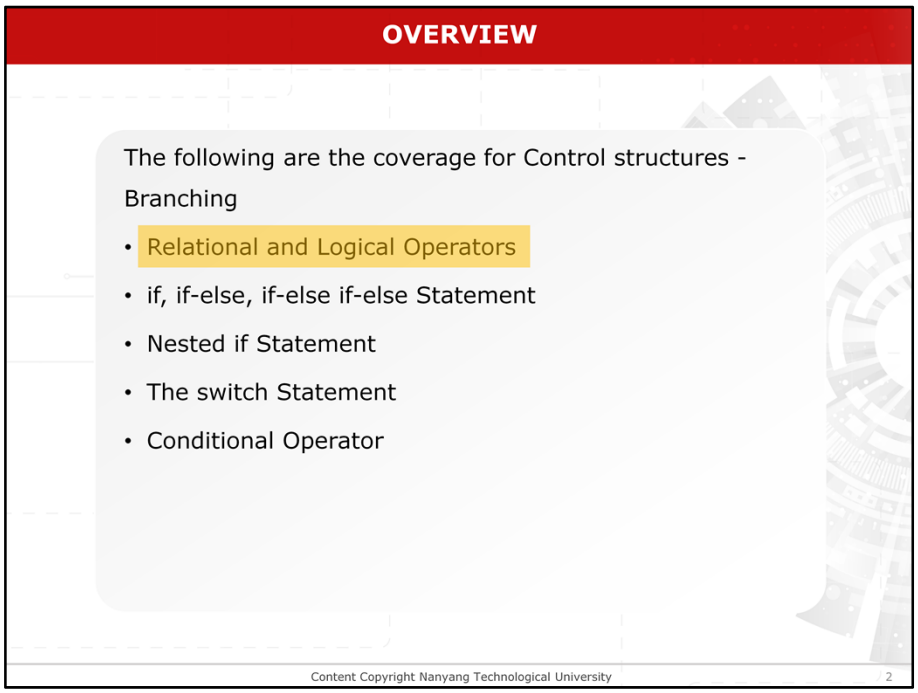


This lesson is on control structure branching



Basic C Programming

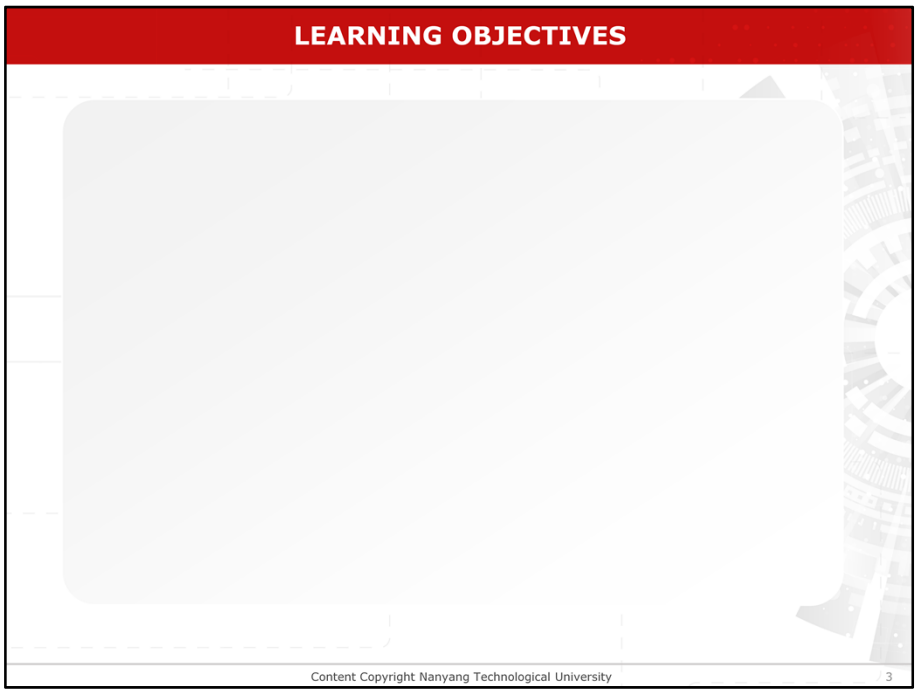
There are 5 main sections to cover for Control structures (branching).

- Relational and Logical Operators
- if, if…else, if….else if….else if….else Statements
- Nested if Statements
- The switch Statements
- Conditional Operator

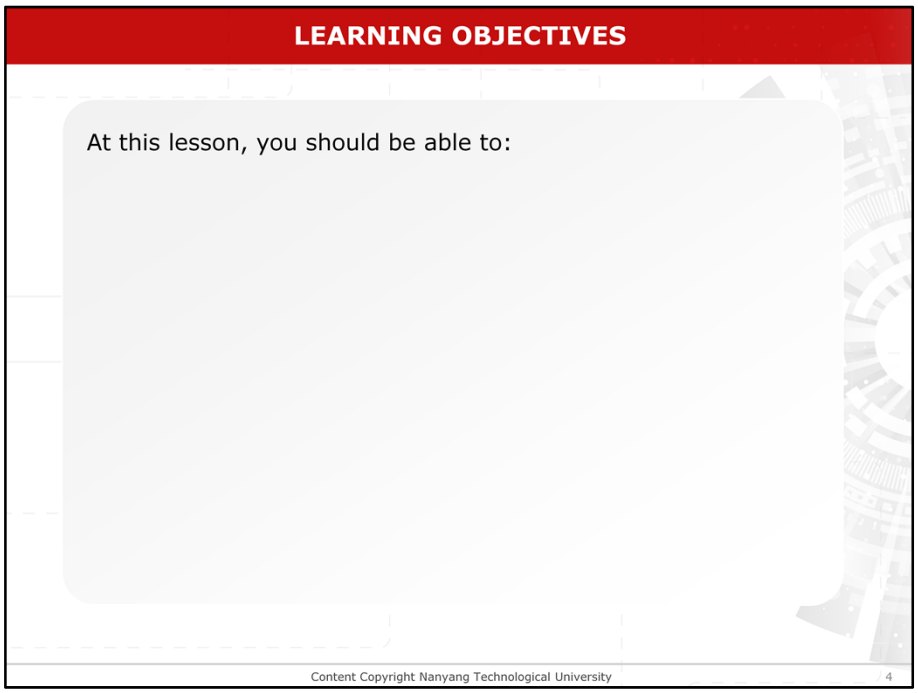
This video lesson focuses on the first part: Relational and logical operators. The execution of C programming statements is normally in sequence from start to end. In the last lecture, we have discussed the simple data types, arithmetic calculations, simple assignment statements and simple input/output. With these statements, we can design simple C programs.

However, the majority of challenging problems requires programs with the ability to make decisions as to what code to execute and the ability to execute certain portions of the code repeatedly. C provides a number of statements that allow branching (or selection) and looping (or repetition).

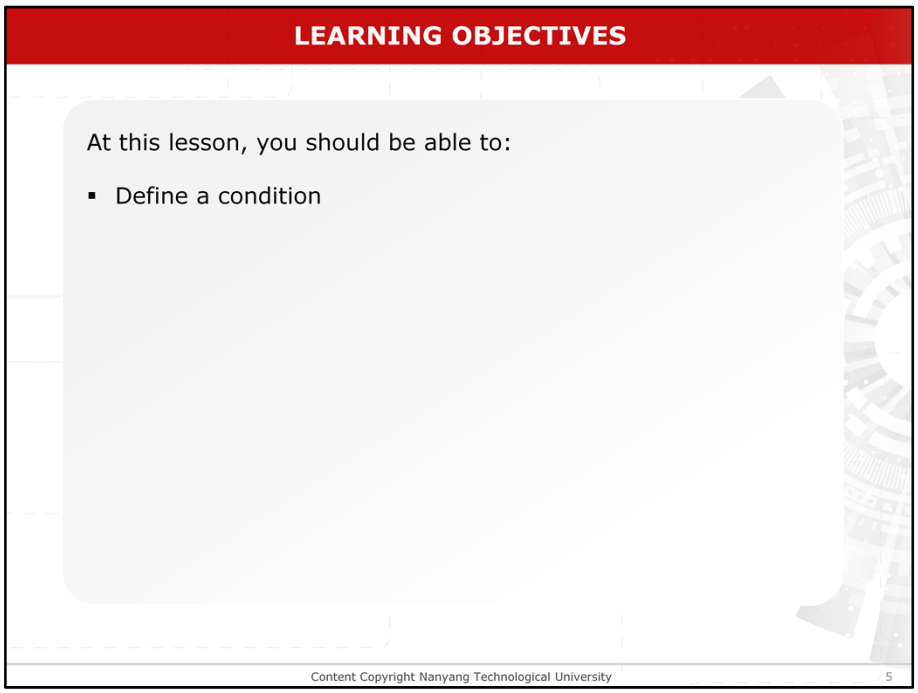
The branching constructs such as the **if-else** statement and the **switch** statement enable us make selection. Another kind of control structure is loop, which allows us execute a group of statements repeatedly for any number of times. This will be discussed later.



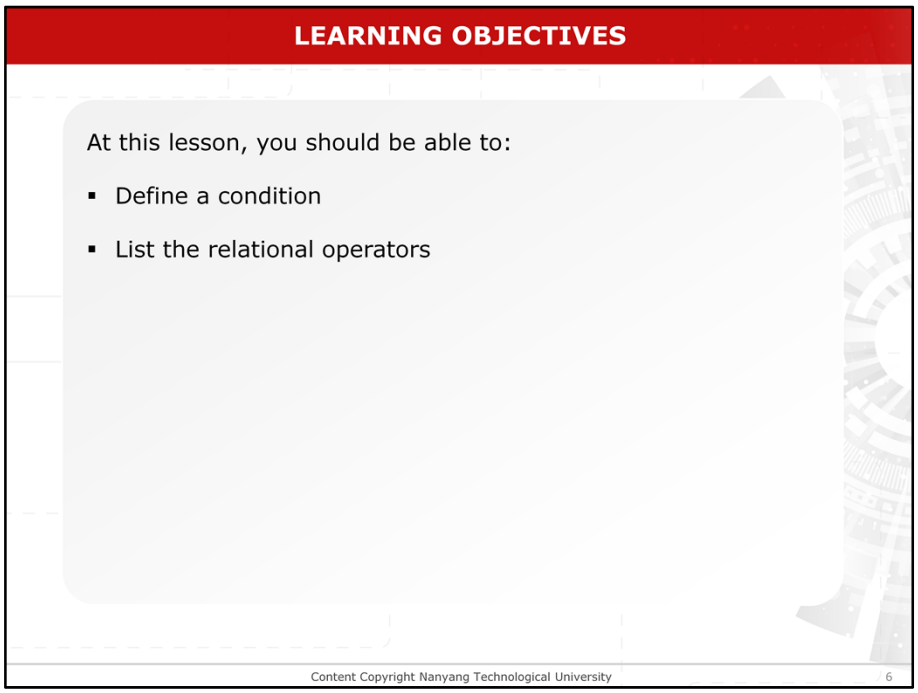
Learrning objectives



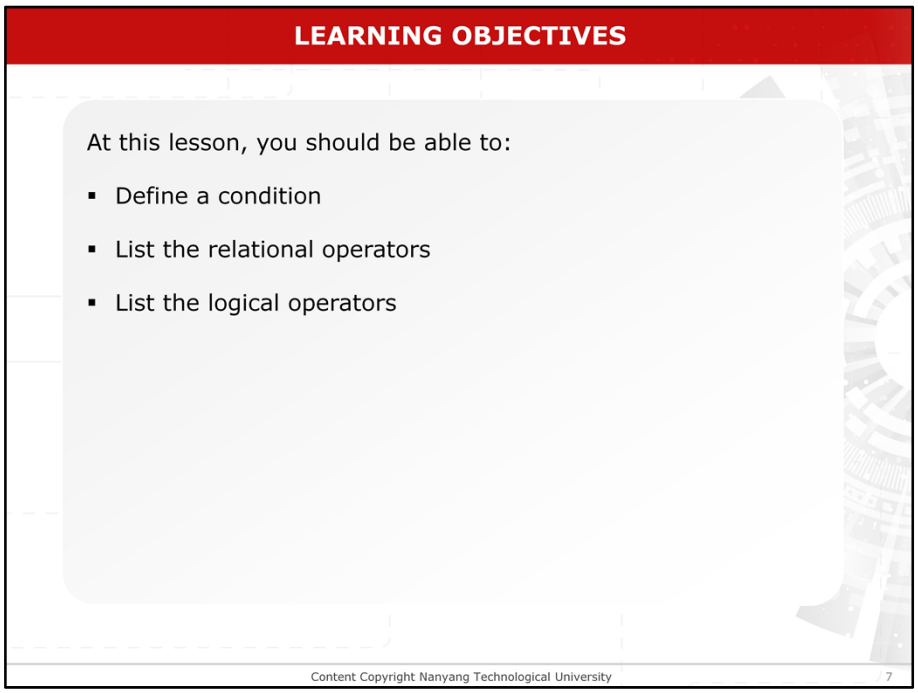
At this lesson, you should be able to:



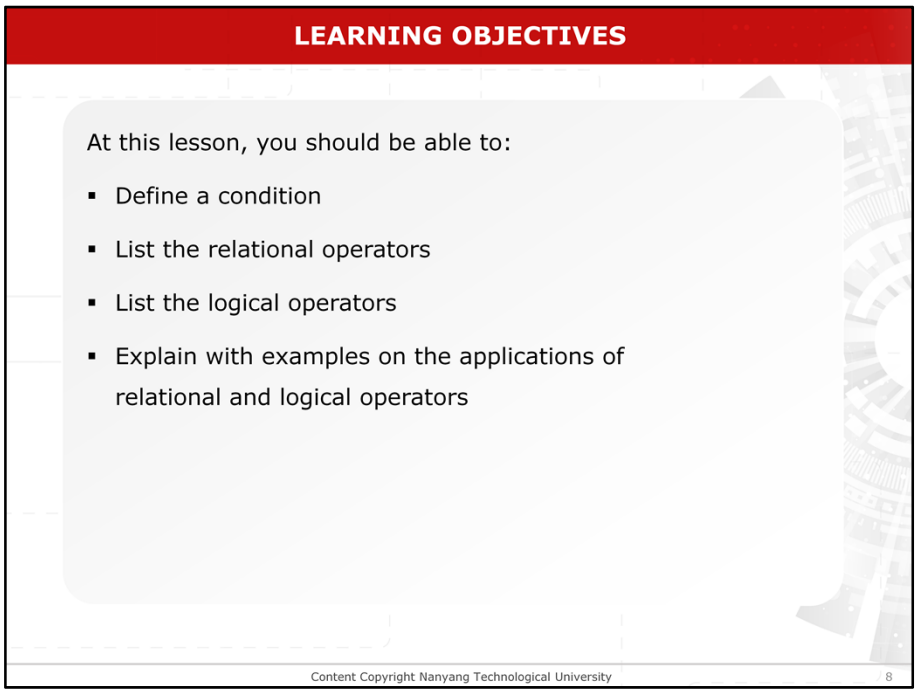
Define a condition



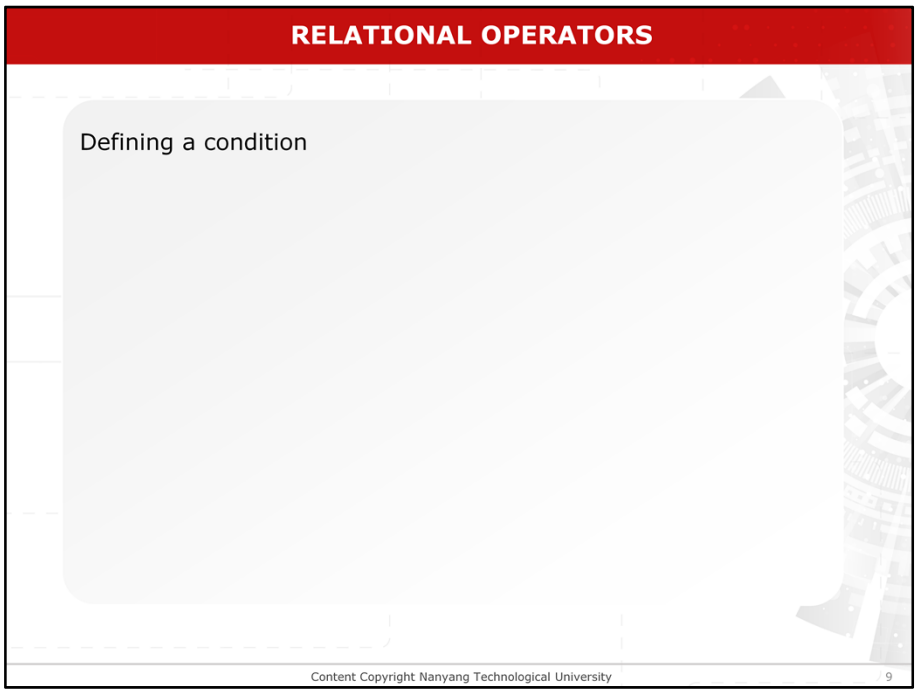
List the relational operators



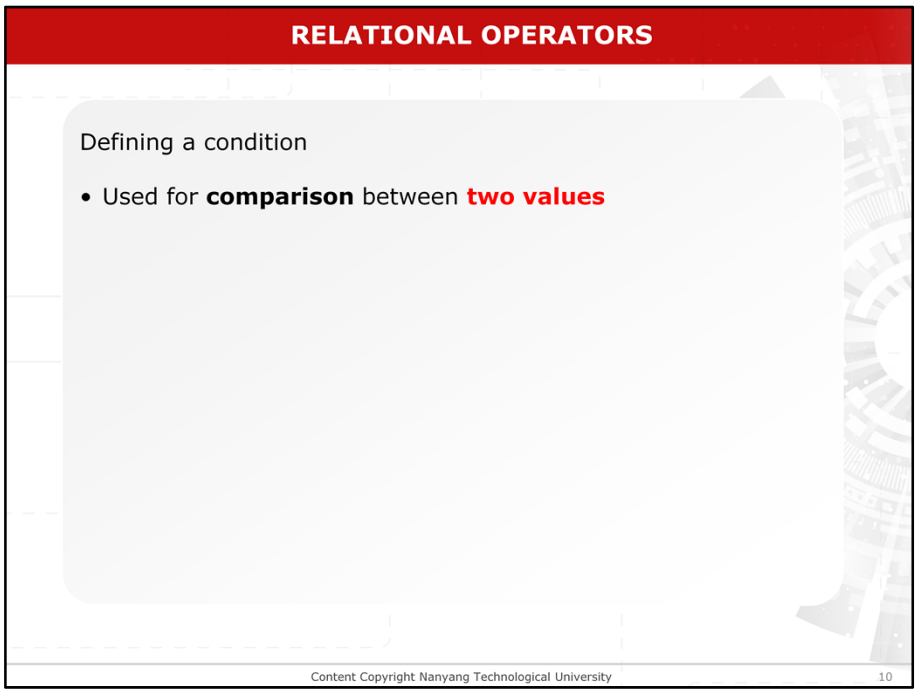
List the logical operators



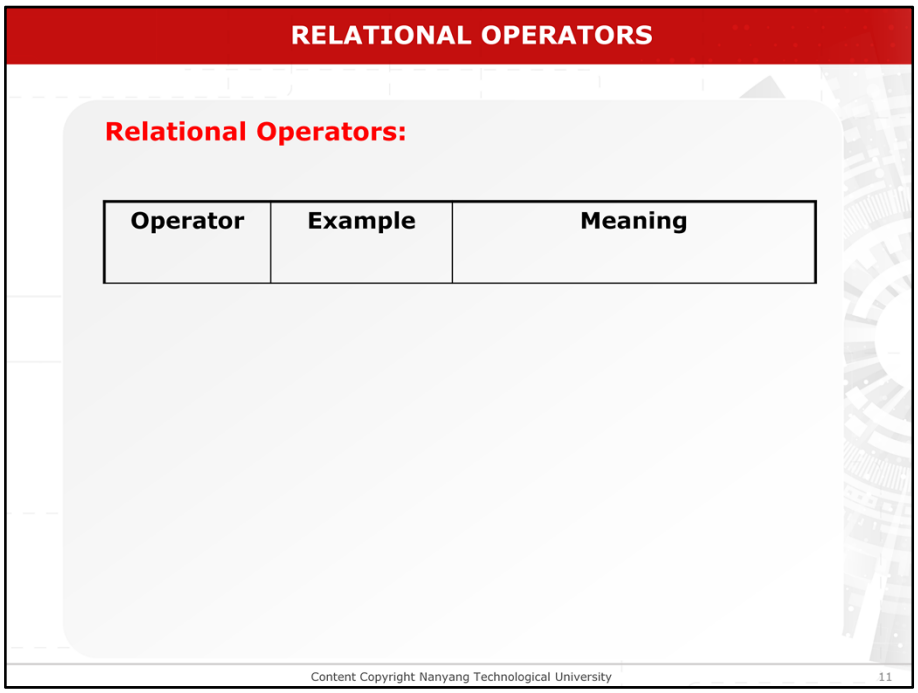
Explain with examples on the applications of relational and logical operators



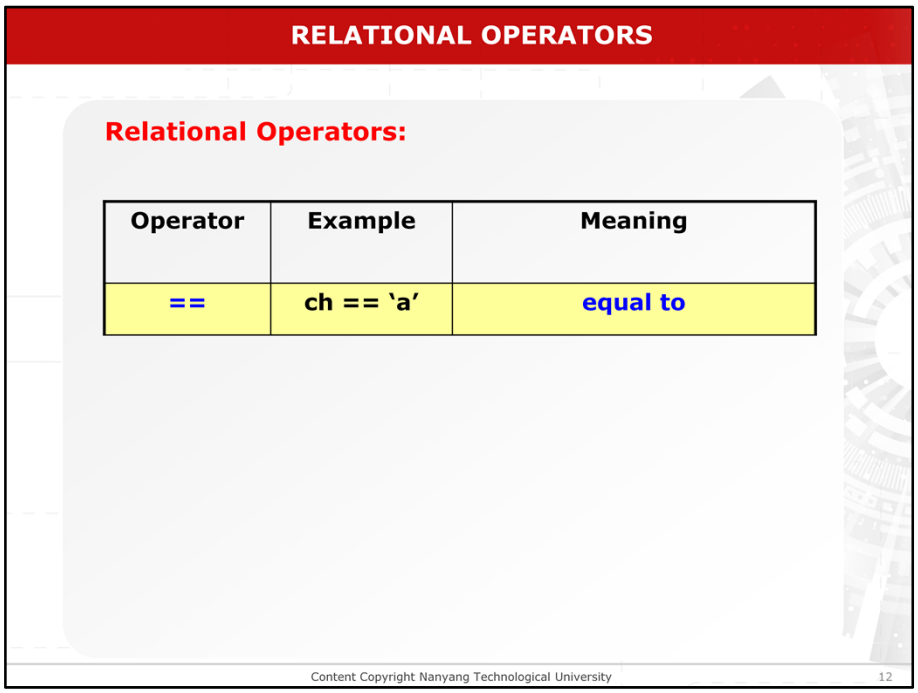
Defining a condition:



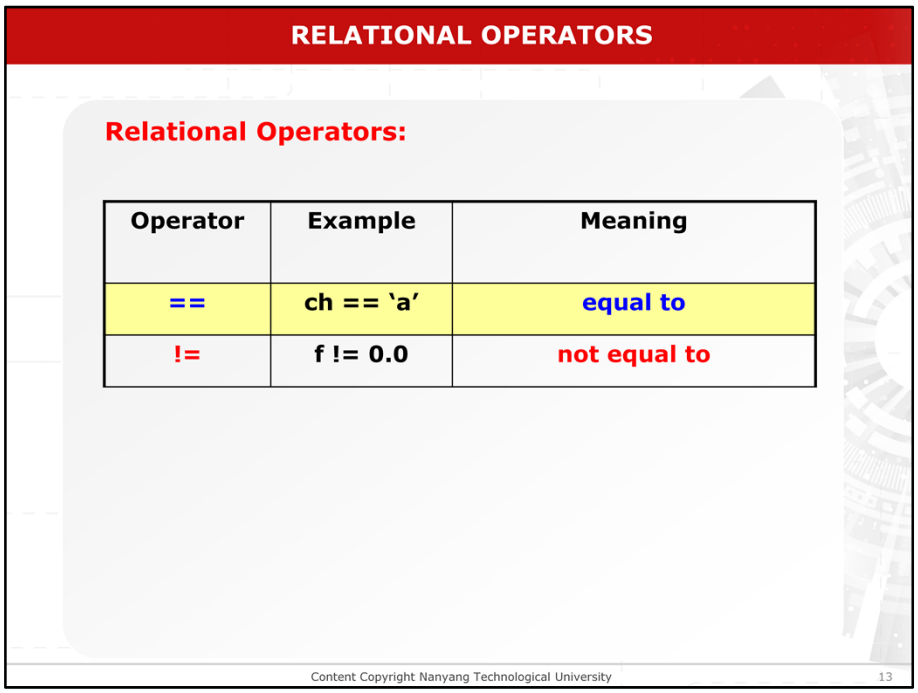
In a branching operation, the decision on which statements to be executed is based on a comparison between two values.



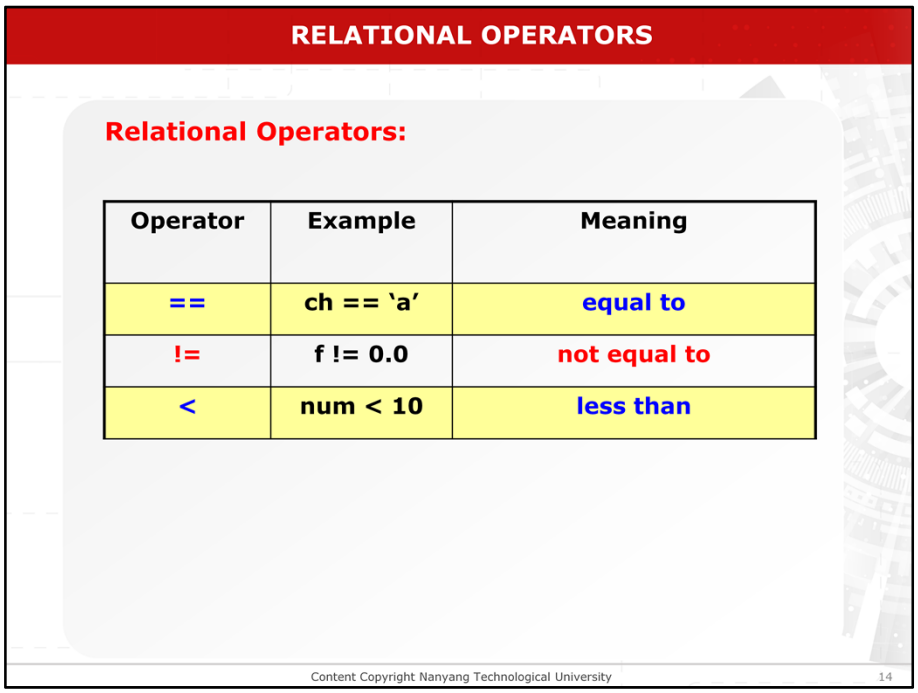
To support this, C provides relational operators. Relational expressions involving relational operators are an essential part of control structures for branching and looping. Relational operators in C include


RELATIONAL OPERATORS
Relational Operators:
Operator
Example
Meaning
==
ch == 'a'
equal to
Content Copyright Nanyang Technological University
12

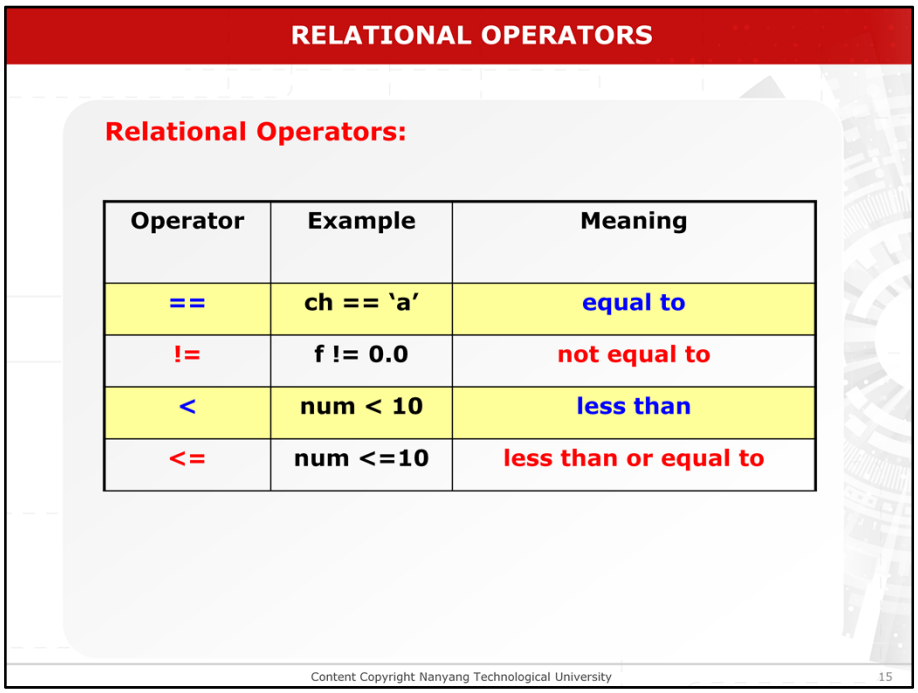
equal to (==)


RELATIONAL OPERATORS
Relational Operators:
Operator
Example
Meaning
==
ch == 'a'
equal to
!=
f != 0.0
not equal to
Content Copyright Nanyang Technological University
13

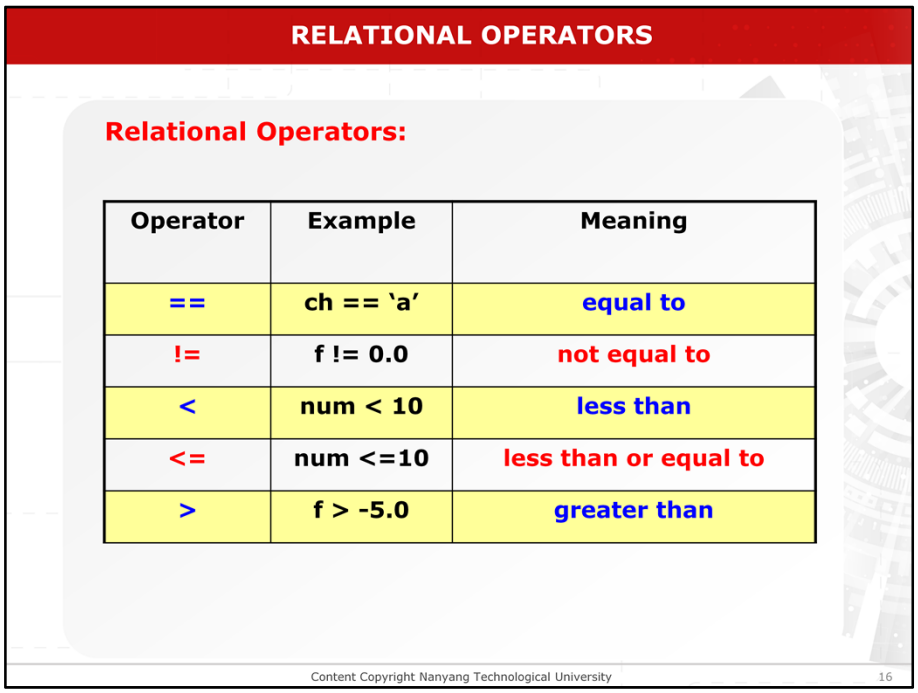
not equal to (!=)


RELATIONAL OPERATORS
Relational Operators:
Operator
Example
Meaning
==
ch == 'a'
equal to
!=
f != 0.0
not equal to
<
num < 10
less than
Content Copyright Nanyang Technological University
14

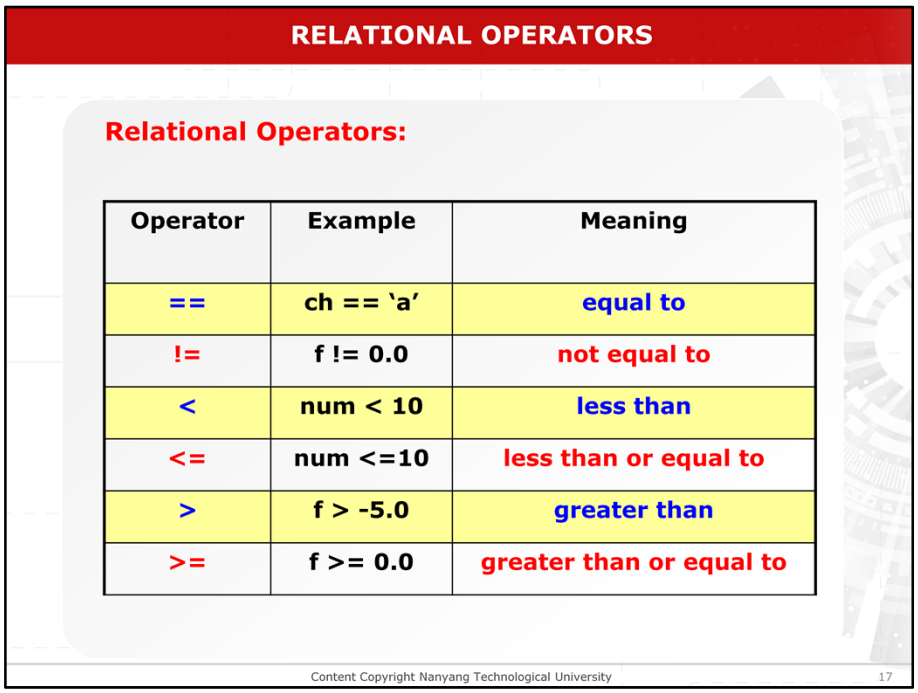
less than (<)



less than or equal to (<=)

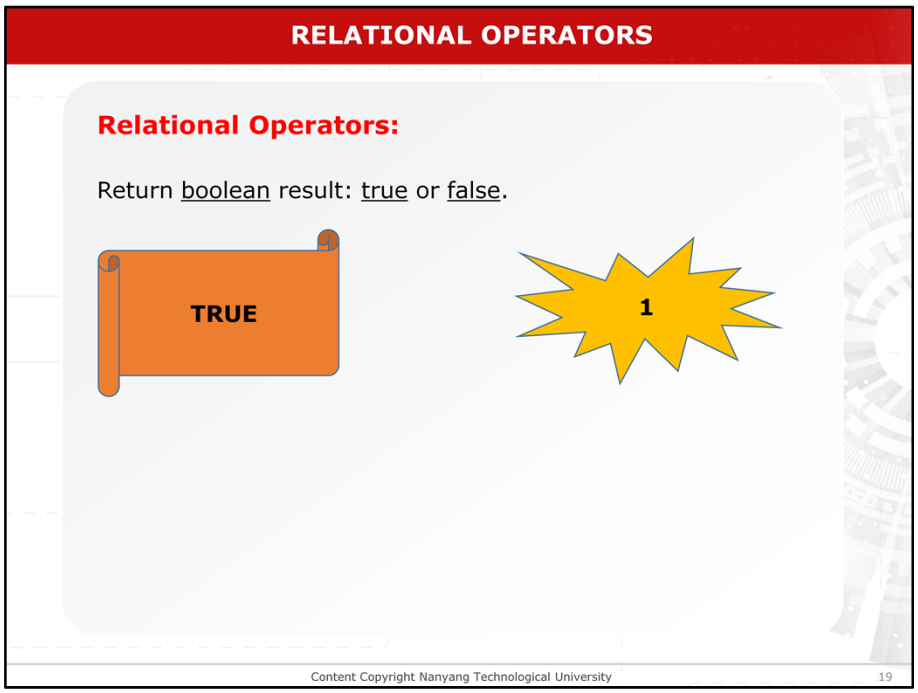


greater than (>)

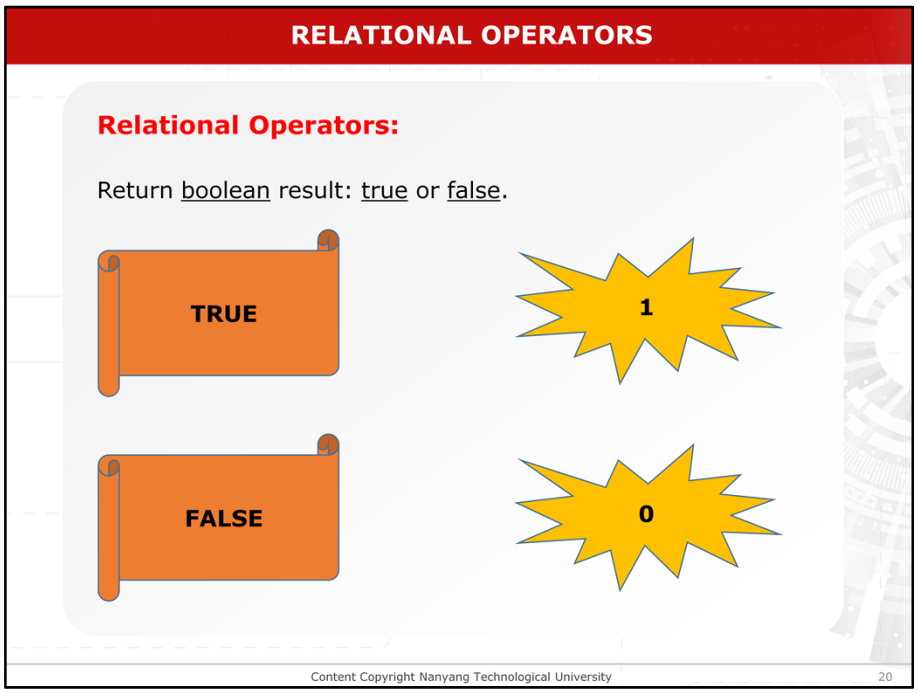


and greater than or equal to (>=).

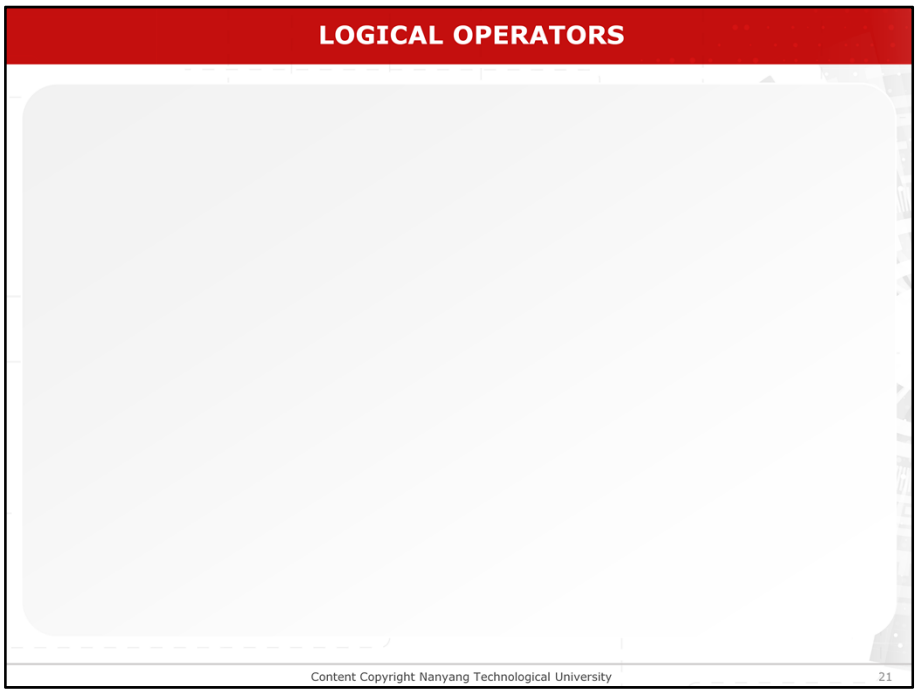
These operators are binary. They perform computations on their operands and return the result as either true or false.



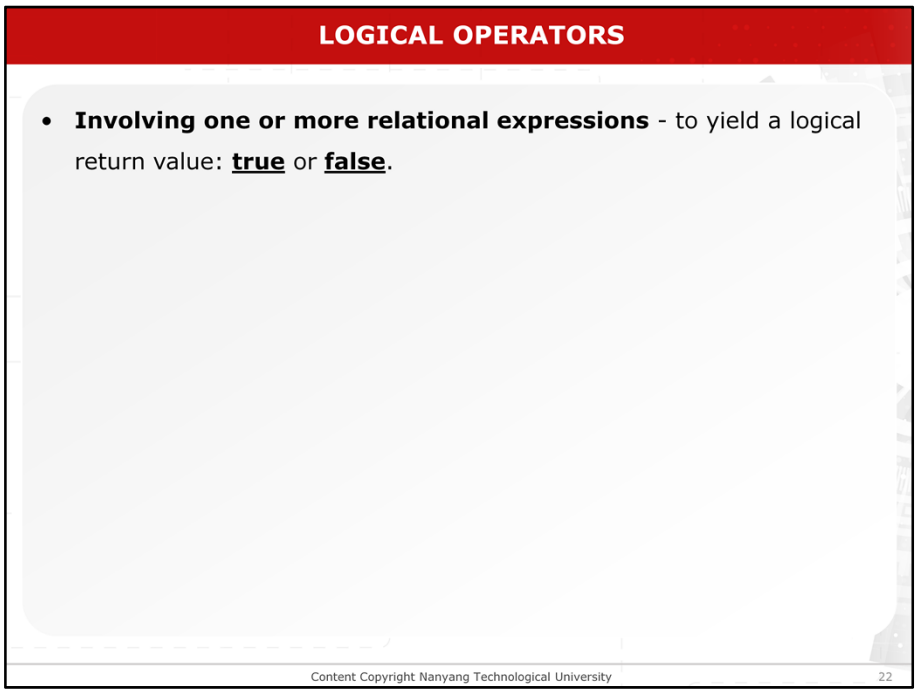
If the result is true, an integer value of 1 is returned,



and if the result is false, then the integer value of 0 is returned.

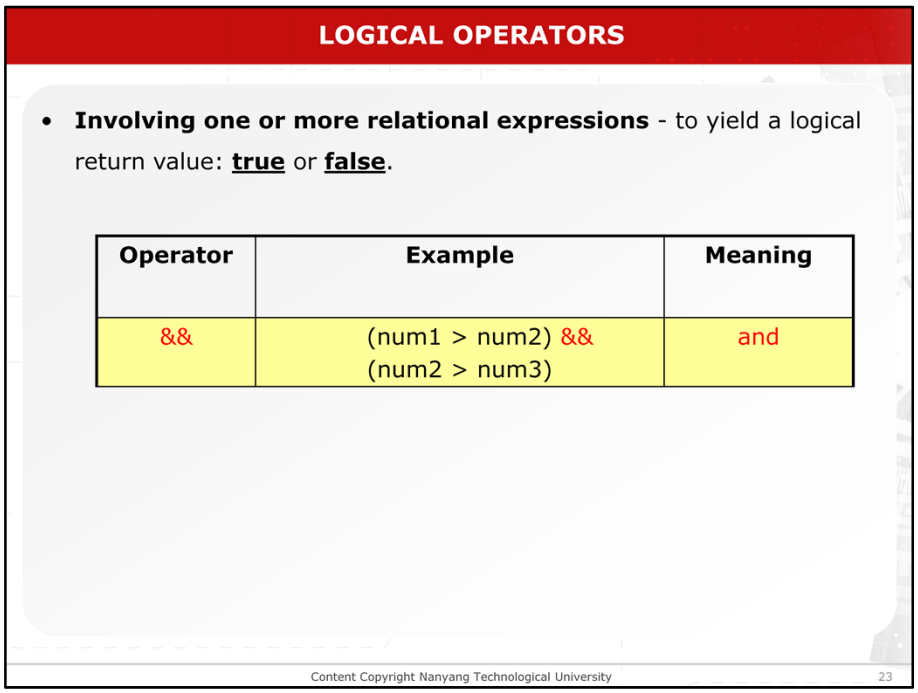


Logical Operators



Logical Operators

Logical operators work on one or more relational expressions to yield either the logical value true or false.



Logical Operators

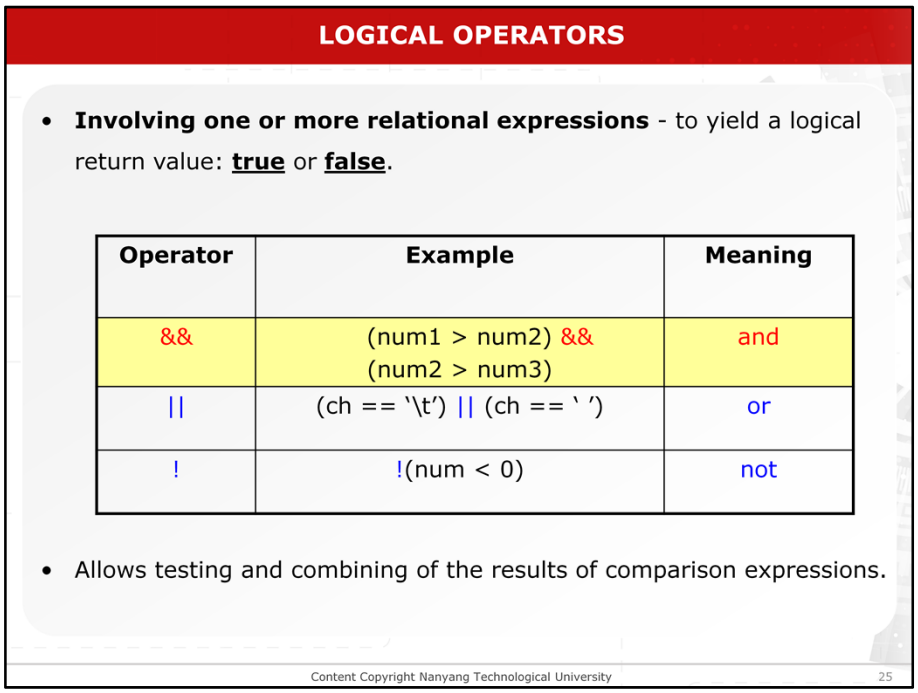
Both logical **and** (**&&)** and logical **or** (**||)** operators are binary operators,

LOGICAL OPERATORS

- **Involving one or more relational expressions** - to yield a logical return value: **true** or **false**.

Operator	Example	Meaning
&&	(num1 > num2) && (num2 > num3)	and
||	(ch == '\t') || (ch == ' ')	or

while the logical **not** operator (**!**) is a unary operator.



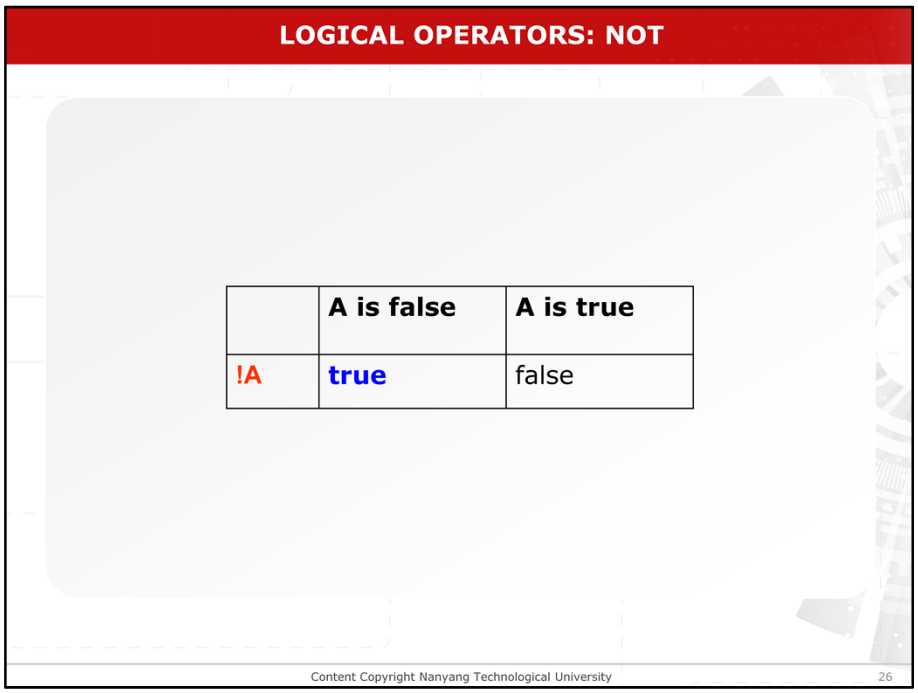
LOGICAL OPERATORS

- **Involving one or more relational expressions** - to yield a logical return value: **true** or **false**.

Operator	Example	Meaning
&&	(num1 > num2) && (num2 > num3)	and
||	(ch == '\t') || (ch == ' ')	or
!	!(num < 0)	not

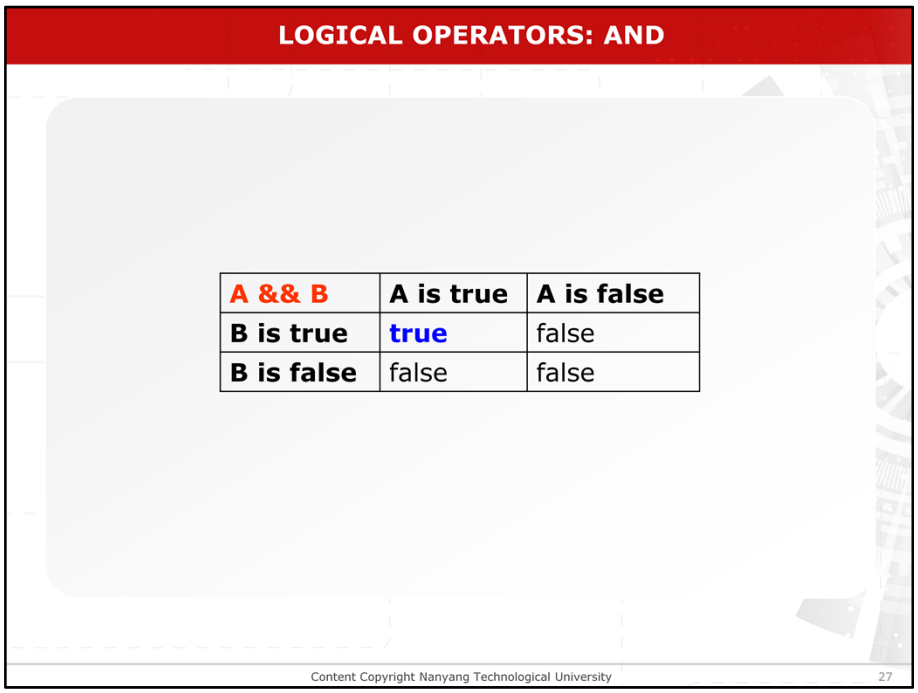
- Allows testing and combining of the results of comparison expressions.

Logical operators allow testing and combining of the results of comparison expressions.



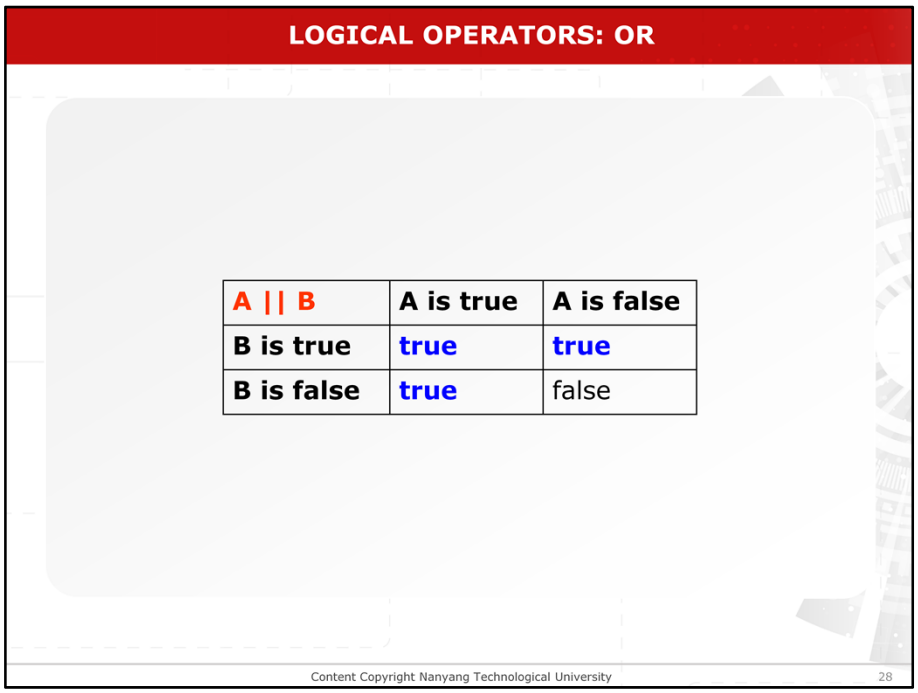
Logical Operators

Logical **not** operator (**!**) returns true when the operand is false and returns false when the operand is true.

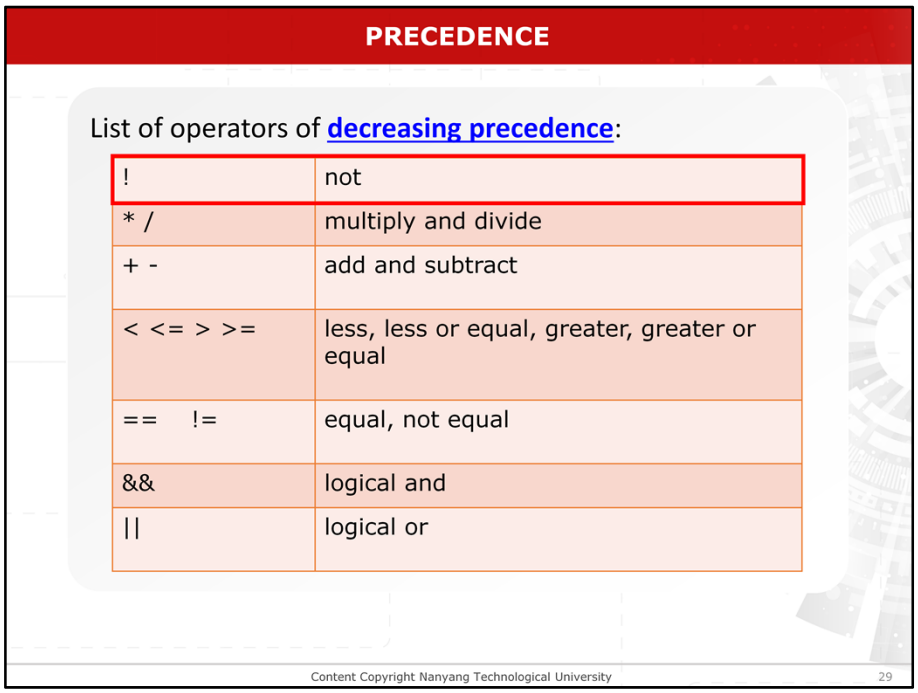


Logical Operators

Logical **and** operator (**&&)** returns true when both operands are true, otherwise it returns false.

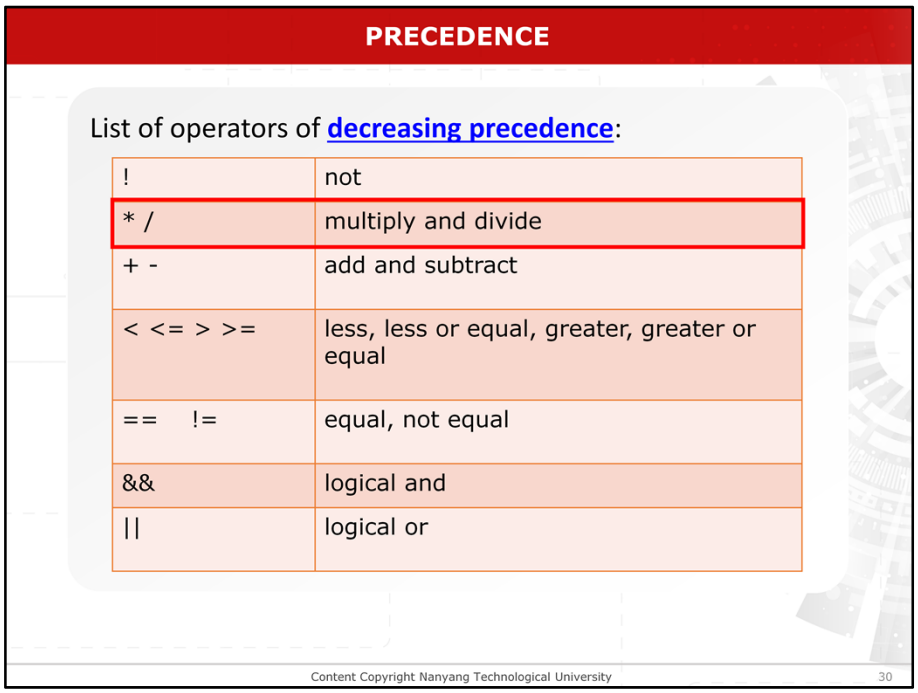


Logical **or** operator (**||**) returns false when both operands are false, otherwise it returns true.



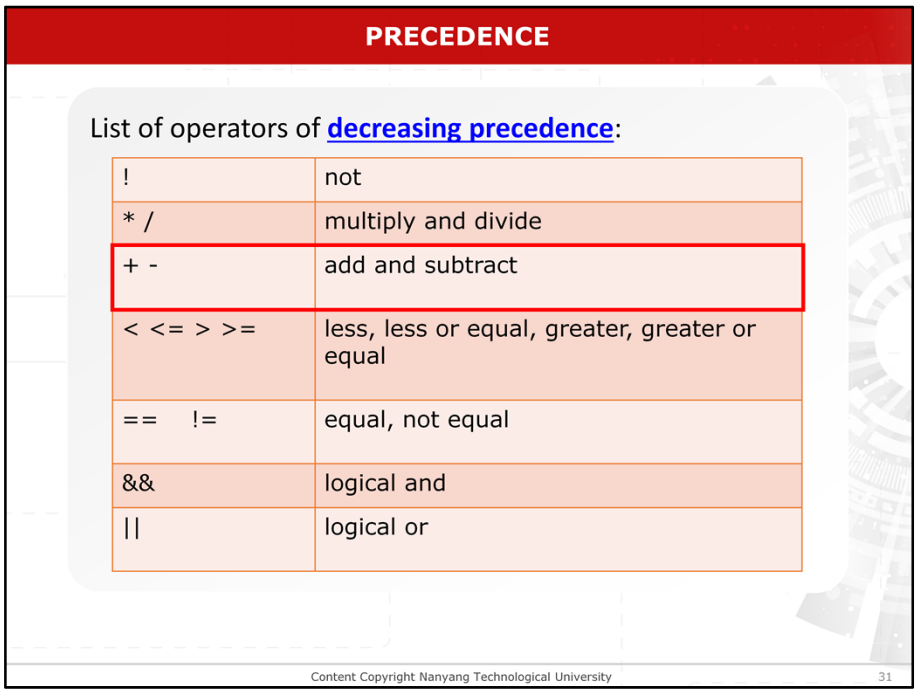
Operator Precedence

The logical **not** (**!**) operator has the highest priority.



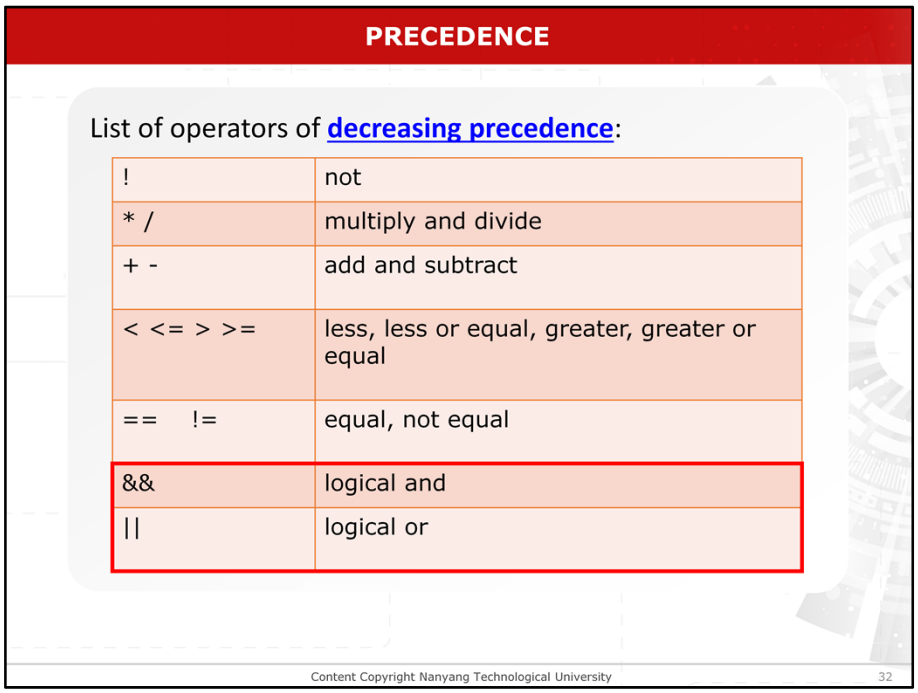
Operator Precedence

It is followed by the multiplication, division



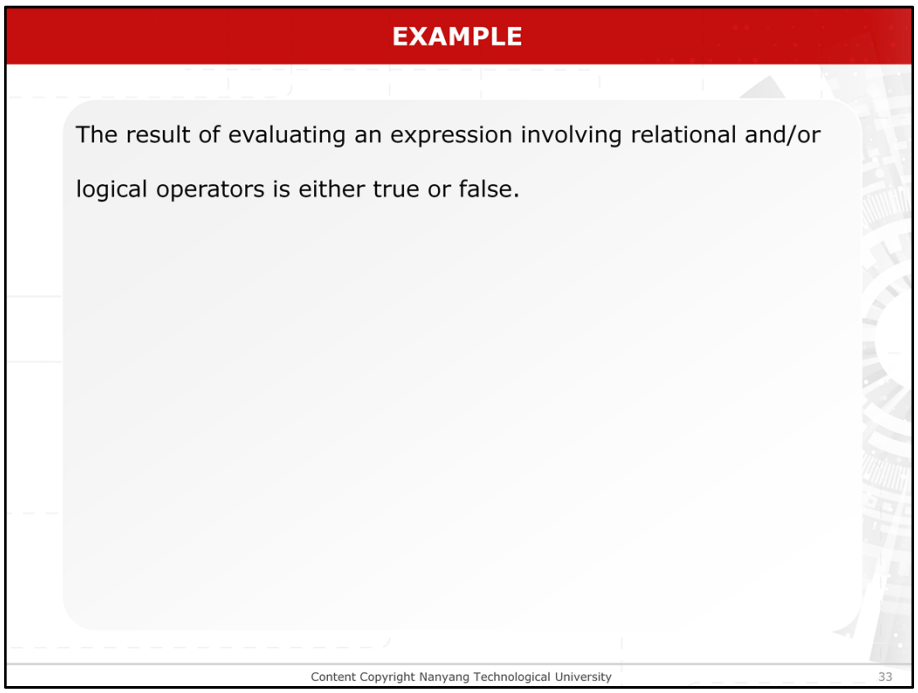
Operator Precedence

FOLLOWED BY addition and subtraction operators.



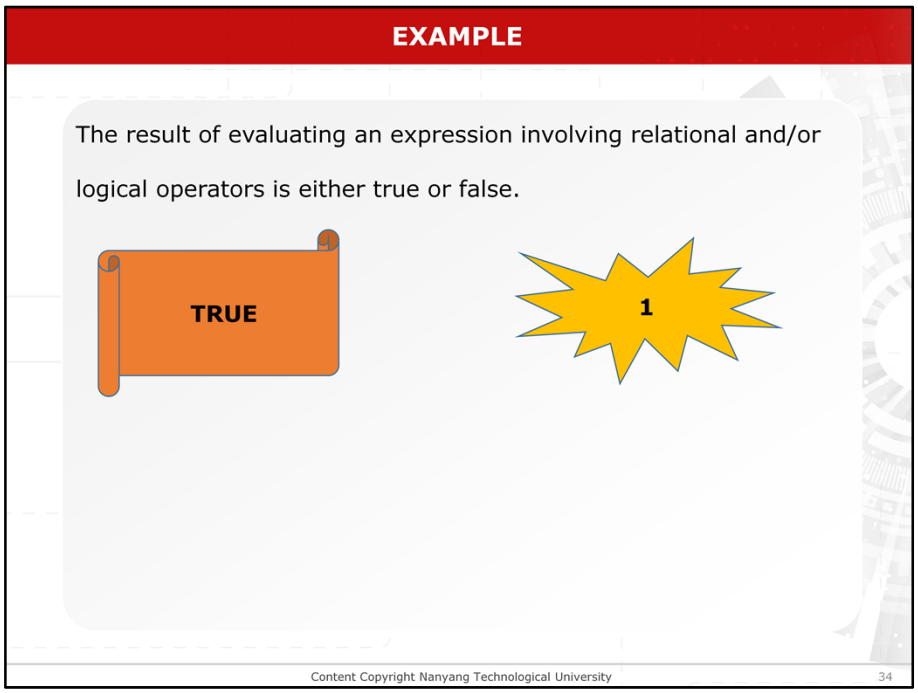
Operator Precedence

The logical **and** (**&&**) and **or** (**||**) operators have a lower priority than the relational operators.

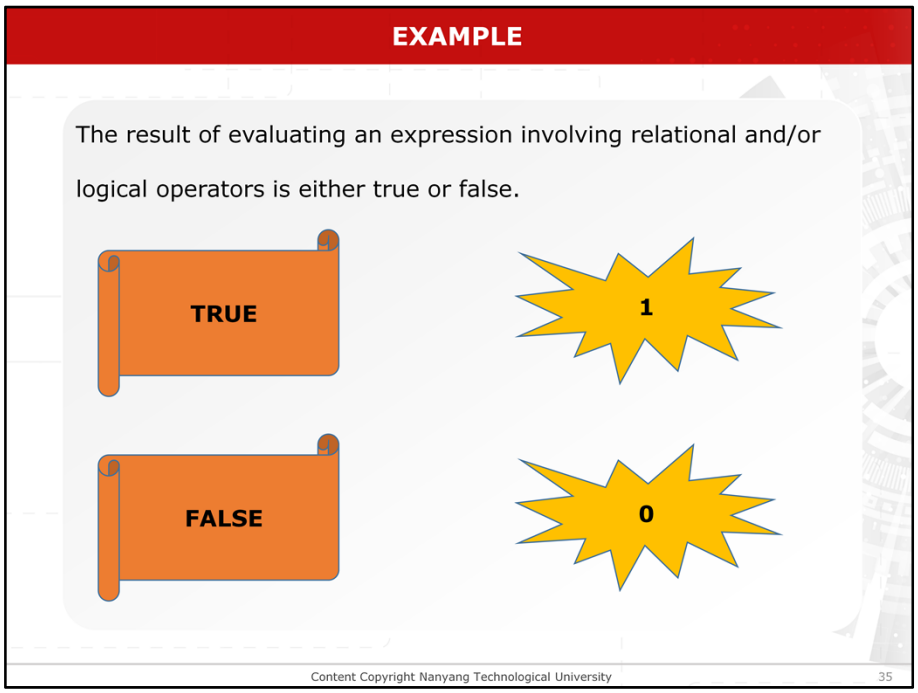


Example

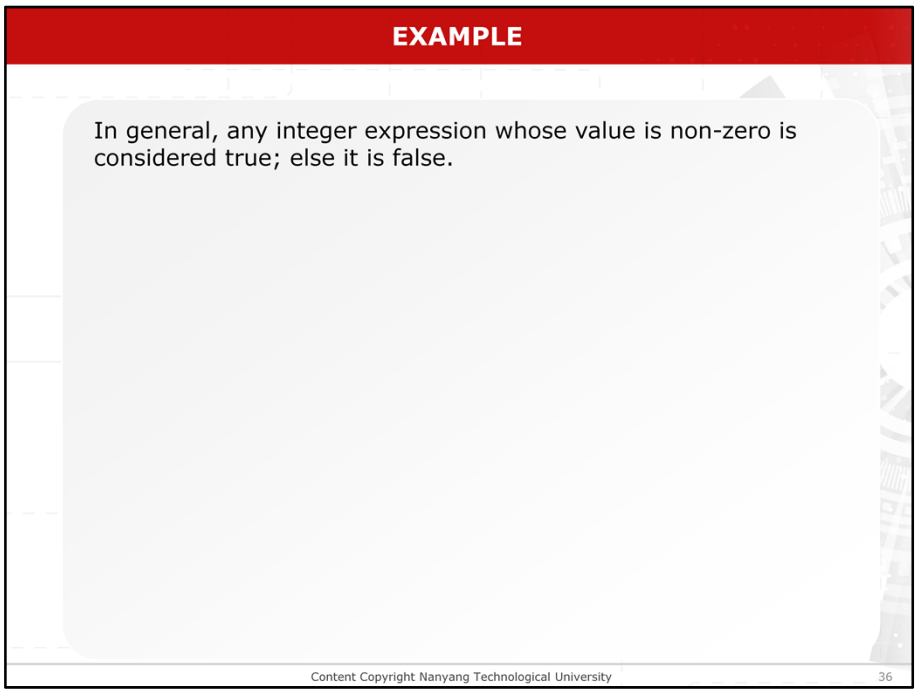
The result of evaluating an expression involving relational and/or logical operators is either true or false.



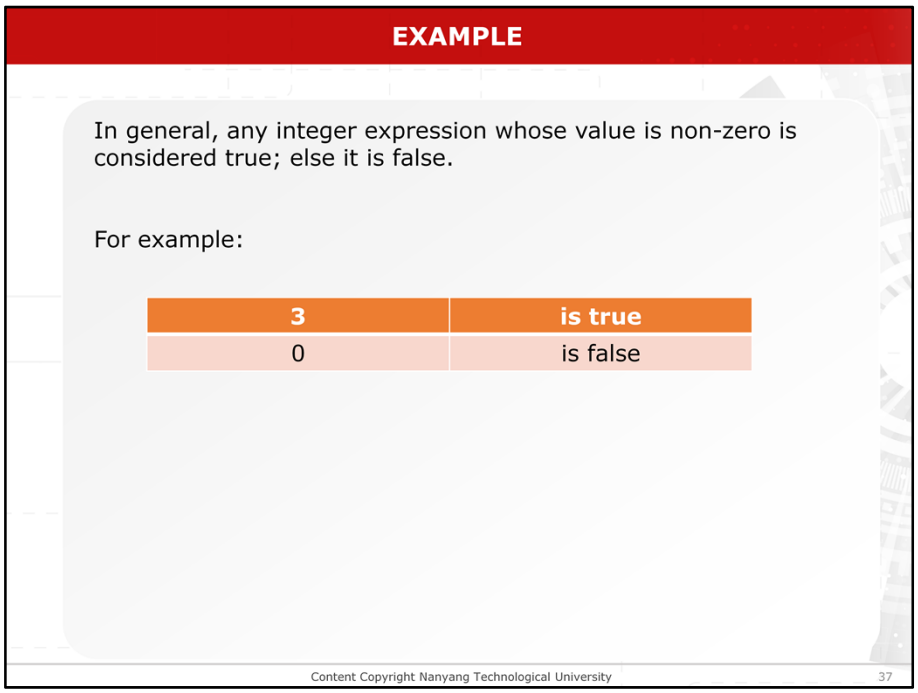
Example
When the result is true, it is 1.



Otherwise it is 0, since C uses 0 to represent a false condition.



Generally, any integer expression whose value is *non-zero* is considered *true*; otherwise it is *false*.



Therefore, 3 is true, and 0 is false. (1 && 0) is false and (1 || 0) is true.

EXAMPLE

In general, any integer expression whose value is non-zero is considered true; else it is false.

For example:

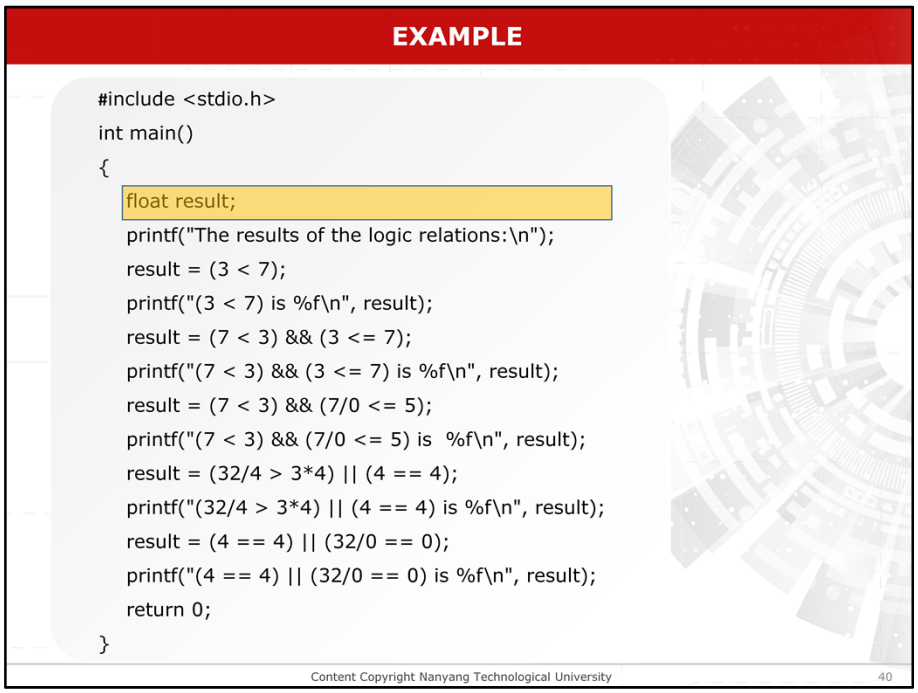
3	is true
0	is false
1 && 0	is false
1 || 0	is true

(1 && 0) is false and (1 || 0) is true.

EXAMPLE

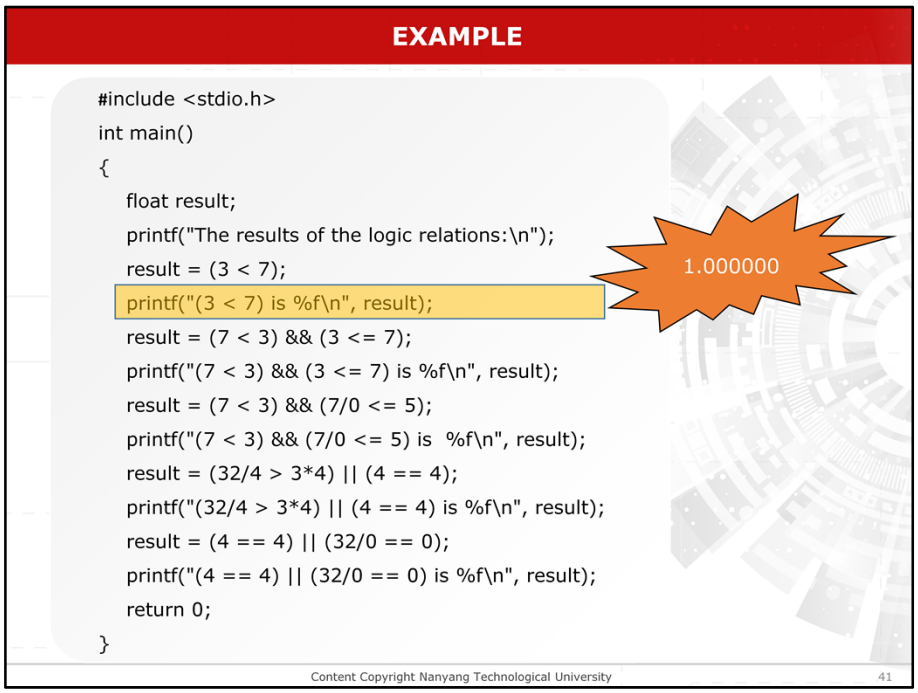
```
#include <stdio.h>
int main()
{
    float result;
    printf("The results of the logic relations:\n");
    result = (3 < 7);
    printf("(3 < 7) is %f\n", result);
    result = (7 < 3) && (3 <= 7);
    printf("(7 < 3) && (3 <= 7) is %f\n", result);
    result = (7 < 3) && (7/0 <= 5);
    printf("(7 < 3) && (7/0 <= 5) is  %f\n", result);
    result = (32/4 > 3*4) || (4 == 4);
    printf("(32/4 > 3*4) || (4 == 4) is %f\n", result);
    result = (4 == 4) || (32/0 == 0);
    printf("(4 == 4) || (32/0 == 0) is %f\n", result);
    return 0;
}
```


Another example is given in the following program to show the logic values of relational and logical expressions.

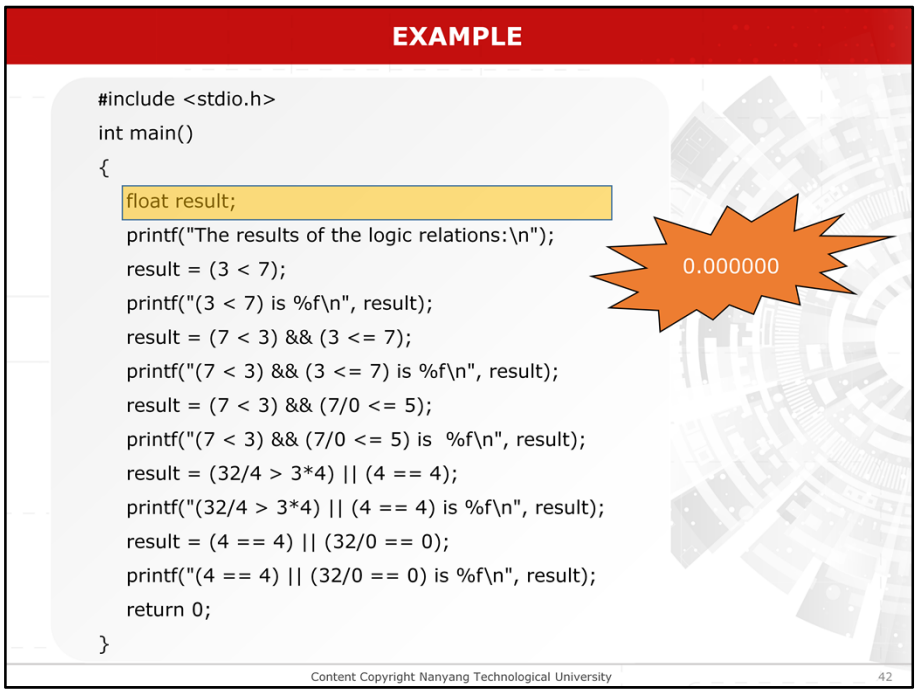


.

The variable **result** is defined as type **float**.



When the variable is printed with the specifier "**%f**", the true value to be printed will be 1.000000



and the false value to be printed will be 0.000000.

EXAMPLE

```
#include <stdio.h>
int main()
{
    float result;
    printf("The results of the logic relations:\n");
    result = (3 < 7);
    printf("(3 < 7) is %f\n", result);
    result = (7 < 3) && (3 <= 7);
    printf("(7 < 3) && (3 <= 7) is %f\n", result);
    result = (7 < 3) && (7/0 <= 5);
    printf("(7 < 3) && (7/0 <= 5) is  %f\n", result);
    result = (32/4 > 3*4) || (4 == 4);
    printf("(32/4 > 3*4) || (4 == 4) is %f\n", result);
    result = (4 == 4) || (32/0 == 0);
    printf("(4 == 4) || (32/0 == 0) is %f\n", result);
    return 0;
}
```


The results are straightforward except the two special cases involving the evaluation of expressions consisting of logical **or** and logical **and** operators.

EXAMPLE

```
#include <stdio.h>
int main()
{
    float result;
    printf("The results of the logic relations:\n");
    result = (3 < 7);
    printf("(3 < 7) is %f\n", result);
    result = (7 < 3) && (3 <= 7);
    printf("(7 < 3) && (3 <= 7) is %f\n", result);
    result = (7 < 3) && (7/0 <= 5);
    printf("(7 < 3) && (7/0 <= 5) is  %f\n", result);
    result = (32/4 > 3*4) || (4 == 4);
    printf("(32/4 > 3*4) || (4 == 4) is %f\n", result);
    result = (4 == 4) || (32/0 == 0);
    printf("(4 == 4) || (32/0 == 0) is %f\n", result);
    return 0;
}
```

 44

First, consider the evaluation of the following statement involving logical **and** operator:

result = (7 < 3) && (7/0 <= 5);

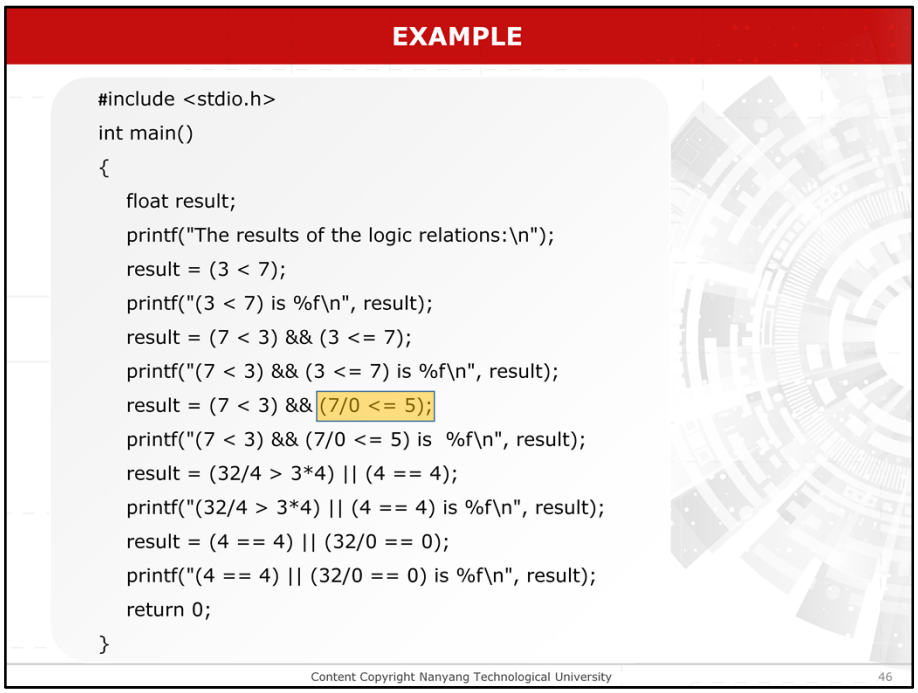
.

EXAMPLE

```
#include <stdio.h>
int main()
{
    float result;
    printf("The results of the logic relations:\n");
    result = (3 < 7);
    printf("(3 < 7) is %f\n", result);
    result = (7 < 3) && (3 <= 7);
    printf("(7 < 3) && (3 <= 7) is %f\n", result);
    result = (7 < 3) && (7/0 <= 5);
    printf("(7 < 3) && (7/0 <= 5) is  %f\n", result);
    result = (32/4 > 3*4) || (4 == 4);
    printf("(32/4 > 3*4) || (4 == 4) is %f\n", result);
    result = (4 == 4) || (32/0 == 0);
    printf("(4 == 4) || (32/0 == 0) is %f\n", result);
    return 0;
}
```

0.000000

When the first relational expression, that is 7 less than 3, is evaluated to be false, the second relational expression does not need to be evaluated.

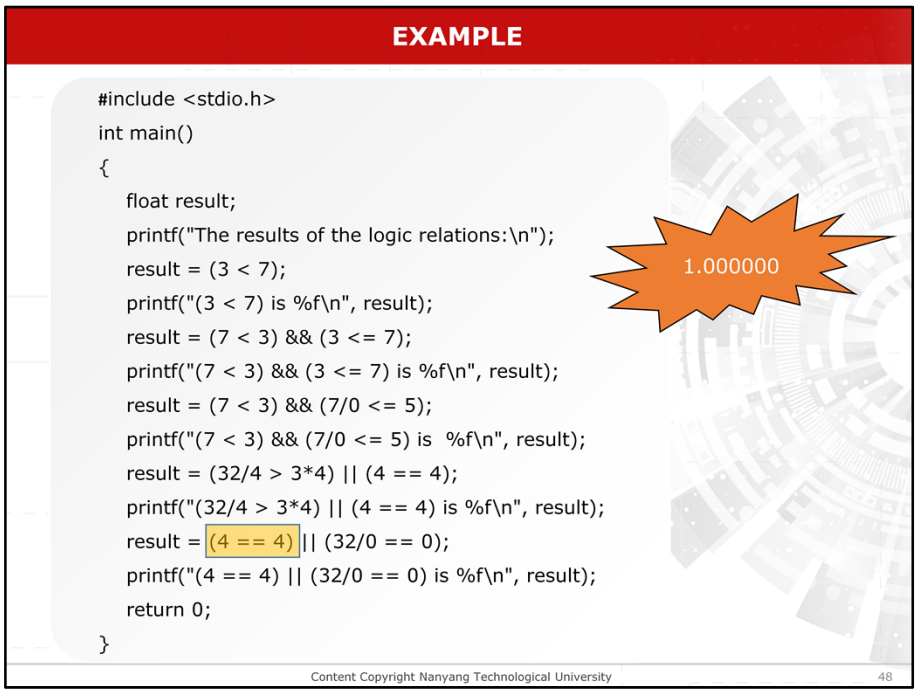


Therefore, even though the second expression contains an error in the evaluation of **7/0**, it does not occur in the overall result. This is called *short-circuit evaluation* in which the second argument is only executed or evaluated if the first argument does not suffice to determine the value of the expression.

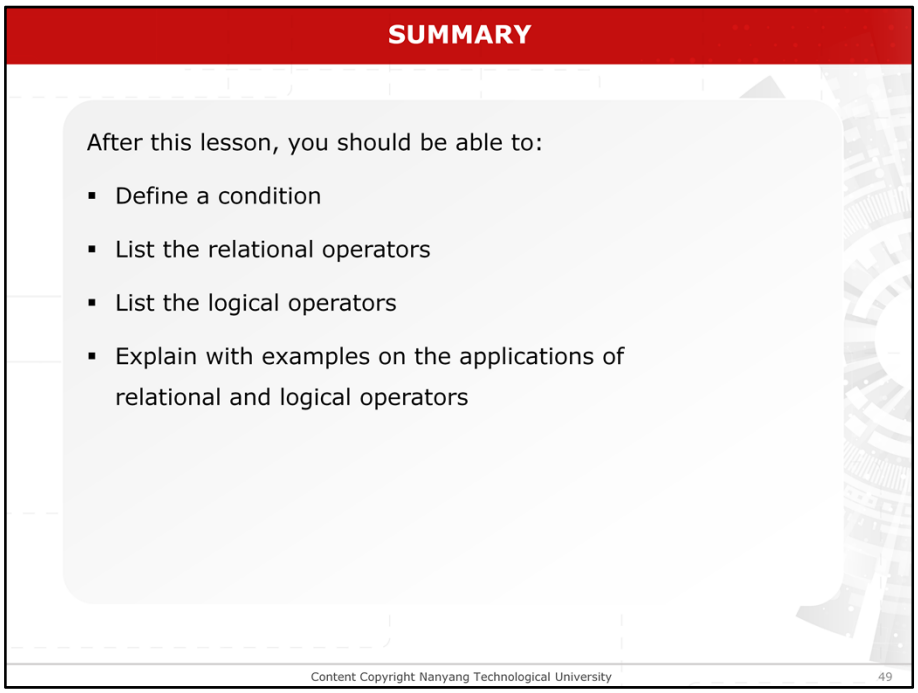
EXAMPLE

```
#include <stdio.h>
int main()
{
    float result;
    printf("The results of the logic relations:\n");
    result = (3 < 7);
    printf("(3 < 7) is %f\n", result);
    result = (7 < 3) && (3 <= 7);
    printf("(7 < 3) && (3 <= 7) is %f\n", result);
    result = (7 < 3) && (7/0 <= 5);
    printf("(7 < 3) && (7/0 <= 5) is  %f\n", result);
    result = (32/4 > 3*4) || (4 == 4);
    printf("(32/4 > 3*4) || (4 == 4) is %f\n", result);
    result = (4 == 4) || (32/0 == 0);
    printf("(4 == 4) || (32/0 == 0) is %f\n", result);
    return 0;
}
```


Another similar case is also occurred during the evaluation of the following statement containing the logical **or** operator:



Since the first expression is evaluated to be true, the second expression does not need to be evaluated. Hence, the error does not occur in the overall result.



In summary, after viewing this video lesson, your should be able to

- Define a condition
- List the relational operators
- List the logical operators
- Explain with examples on the applications of relational and logical operators