

This lesson is on control structure branching

OVERVIEW

The following are the coverage for Control structures -

Branching

- Relational and Logical Operators
- if, if-else, if-else if-else Statement
- Nested if Statement
- The switch Statement
- Conditional Operator

Content Copyright Nanyang Technological University

2

Basic C Programming

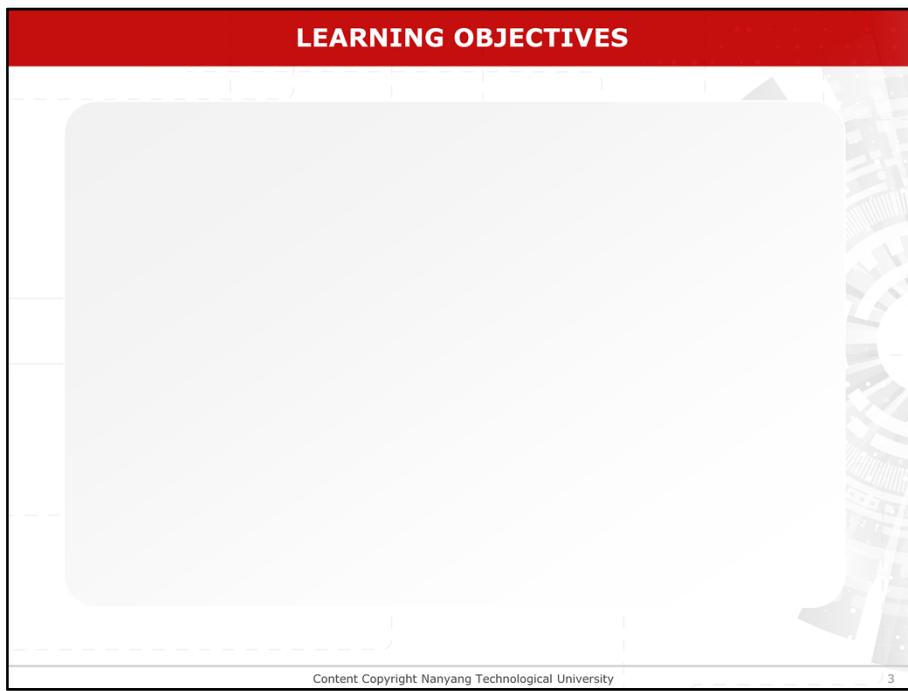
There are 5 main sections to cover for Control structures (branching).

- 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

LEARNING OBJECTIVES

At this lesson, you should be able to:

Content Copyright Nanyang Technological University 4

At this lesson, you should be able to:

LEARNING OBJECTIVES

At this lesson, you should be able to:

- Define a condition

Content Copyright Nanyang Technological University

5

Define a condition

LEARNING OBJECTIVES

At this lesson, you should be able to:

- Define a condition
- List the relational operators

Content Copyright Nanyang Technological University 6

List the relational operators

LEARNING OBJECTIVES

At this lesson, you should be able to:

- Define a condition
- List the relational operators
- List the logical operators

Content Copyright Nanyang Technological University 7

List the logical operators

LEARNING 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

Content Copyright Nanyang Technological University 8

Explain with examples on the applications of relational and logical operators

RELATIONAL OPERATORS

Defining a condition

Content Copyright Nanyang Technological University 9

Defining a condition:

RELATIONAL OPERATORS

Defining a condition

- Used for **comparison** between **two values**

Content Copyright Nanyang Technological University 10

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

The slide has a red header bar with the title 'RELATIONAL OPERATORS'. Below the header is a table with three columns: 'Operator', 'Example', and 'Meaning'. The table is currently empty. At the bottom left of the slide, there is a small text 'Content Copyright Nanyang Technological University' and at the bottom right, the number '11'.

Operator	Example	Meaning

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
==	<code>ch == 'a'</code>	equal to
!=	<code>f != 0.0</code>	not equal to

Content Copyright Nanyang Technological University

13

not equal to (!=)

RELATIONAL OPERATORS

Relational Operators:

Operator	Example	Meaning
==	<code>ch == 'a'</code>	equal to
!=	<code>f != 0.0</code>	not equal to
<	<code>num < 10</code>	less than

Content Copyright Nanyang Technological University

14

less than (<)

RELATIONAL OPERATORS		
Relational Operators:		
Operator	Example	Meaning
==	ch == 'a'	equal to
!=	f != 0.0	not equal to
<	num < 10	less than
<=	num <=10	less than or equal to

less than or equal to (\leq)

RELATIONAL OPERATORS

Relational Operators:

Operator	Example	Meaning
==	<code>ch == 'a'</code>	equal to
!=	<code>f != 0.0</code>	not equal to
<	<code>num < 10</code>	less than
<=	<code>num <=10</code>	less than or equal to
>	<code>f > -5.0</code>	greater than

Content Copyright Nanyang Technological University

16

greater than (>)

RELATIONAL OPERATORS		
Relational Operators:		
Operator	Example	Meaning
==	ch == 'a'	equal to
!=	f != 0.0	not equal to
<	num < 10	less than
<=	num <=10	less than or equal to
>	f > -5.0	greater than
>=	f >= 0.0	greater than or equal to

Content Copyright Nanyang Technological University

17

and greater than or equal to (\geq).

RELATIONAL OPERATORS

Relational Operators:

Return boolean result: true or false.

Content Copyright Nanyang Technological University 18

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

RELATIONAL OPERATORS

Relational Operators:

Return boolean result: true or false.

Content Copyright Nanyang Technological University

19

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

RELATIONAL OPERATORS

Relational Operators:

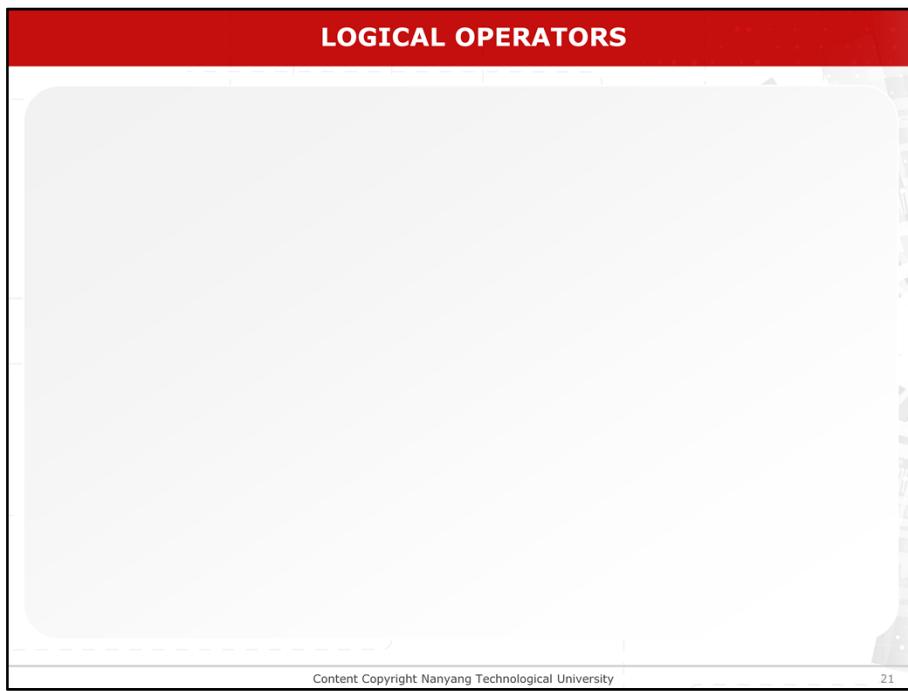
Return boolean result: true or false.

The diagram illustrates the relationship between boolean values and their integer representations. It features two orange scroll-like shapes. The top scroll contains the word "TRUE" and the bottom scroll contains the word "FALSE". To the right of each scroll is a yellow starburst shape. The top starburst contains the number "1" and the bottom one contains the number "0". This visualizes how relational operators like equality (==) and inequality (!=) return boolean values, which are then converted to integers (1 for true, 0 for false).

Content Copyright Nanyang Technological University

20

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



Logical Operators

LOGICAL OPERATORS

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

Content Copyright Nanyang Technological University 22

Logical Operators

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

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

Content Copyright Nanyang Technological University

23

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

Content Copyright Nanyang Technological University

24

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.

Content Copyright Nanyang Technological University

25

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

The slide has a red header bar with the title "LOGICAL OPERATORS: NOT". Below the header is a truth table:

	A is false	A is true
!A	true	false

At the bottom left of the slide, it says "Content Copyright Nanyang Technological University" and at the bottom right it says "26".

Logical Operators

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

LOGICAL OPERATORS: AND

A && B	A is true	A is false
B is true	true	false
B is false	false	false

Content Copyright Nanyang Technological University 27

Logical Operators

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

LOGICAL OPERATORS: OR

A B	A is true	A is false
B is true	true	true
B is false	true	false

Content Copyright Nanyang Technological University 28

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

PRECEDENCE	
List of operators of <u>decreasing precedence</u> :	
!	not
* /	multiply and divide
+ -	add and subtract
< <= > >=	less, less or equal, greater, greater or equal
== !=	equal, not equal
&&	logical and
	logical or

Content Copyright Nanyang Technological University

29

Operator Precedence

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

PRECEDENCE	
List of operators of <u>decreasing precedence</u> :	
!	not
* /	multiply and divide
+ -	add and subtract
< <= > >=	less, less or equal, greater, greater or equal
== !=	equal, not equal
&&	logical and
	logical or

Content Copyright Nanyang Technological University

30

Operator Precedence

It is followed by the multiplication, division

PRECEDENCE	
List of operators of decreasing precedence :	
!	not
* /	multiply and divide
+ -	add and subtract
< <= > >=	less, less or equal, greater, greater or equal
== !=	equal, not equal
&&	logical and
	logical or

Content Copyright Nanyang Technological University

31

Operator Precedence

FOLLOWED BY addition and subtraction operators.

PRECEDENCE	
List of operators of decreasing precedence :	
!	not
* /	multiply and divide
+ -	add and subtract
< <= > >=	less, less or equal, greater, greater or equal
== !=	equal, not equal
&&	logical and
	logical or

Content Copyright Nanyang Technological University

32

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

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

EXAMPLE

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

Content Copyright Nanyang Technological University

34

Example

When the result is true, it is 1.

EXAMPLE

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

The diagram illustrates the relationship between boolean values and their binary representations. It features two orange scroll-like boxes. The top box contains the word "TRUE" and the bottom box contains the word "FALSE". To the right of each box is a yellow starburst shape containing the digit "1" above the "TRUE" box and the digit "0" below the "FALSE" box. This visualizes that in C, a true condition is represented by the value 1 and a false condition by the value 0.

Content Copyright Nanyang Technological University

35

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

EXAMPLE

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

Content Copyright Nanyang Technological University

36

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

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

Content Copyright Nanyang Technological University

37

Therefore, 3 is true, and 0 is false. $(1 \&\& 0)$ is false and $(1 \mid\mid 0)$ 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

Content Copyright Nanyang Technological University

38

$(1 \&\& 0)$ 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;
}
```

Content Copyright Nanyang Technological University

39

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

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;
}
```

Content Copyright Nanyang Technological University

40

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

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); // This line is highlighted with a yellow box
    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;
}
```

1.000000

Content Copyright Nanyang Technological University

41

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

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;
}
```

 0.000000Content Copyright Nanyang Technological University42

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;
}
```

Content Copyright Nanyang Technological University

43

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;
}
```

Content Copyright Nanyang Technological University

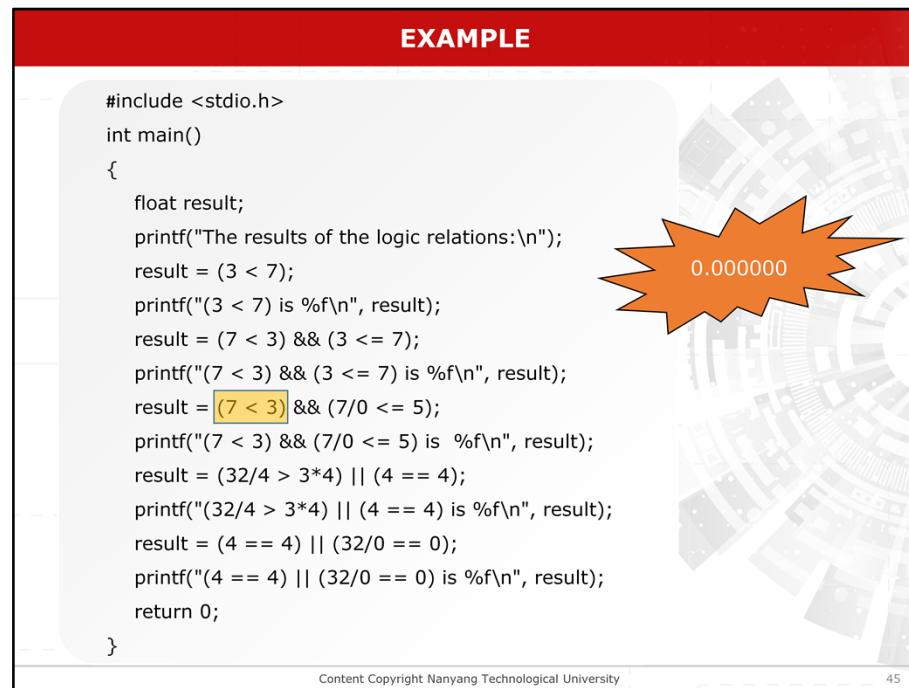
44

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;
}
```



Content Copyright Nanyang Technological University

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.

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;
}
```

Content Copyright Nanyang Technological University

46

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;
}
```

Content Copyright Nanyang Technological University

47

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

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;
}
```

1.000000

Content Copyright Nanyang Technological University

48

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.

SUMMARY

After 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

Content Copyright Nanyang Technological University

49

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