Basics of programming in C++

Prepared by Enas Naffar for the use of programming fundamentals(1) course at Philadelphia University.

The slides include material from Introduction to C++ lecture notes- Massachusetts Institute of Technology and PF1 lecture notes - Philadelphia University.

C++ brief introduction

- C++ is popular, particularly for applications that require speed and/or access to some low-level features. It was created in 1979.
- C++ is a high-level language:
  - when you write a program in it, you don’t need to worry about the details of processor instructions.
  - C++ does give access to some lower-level functionality than other languages (e.g. memory addresses).

First C++ Example

```
// Hello world example
#include <iostream>
int main()
{
    std::cout << "Hello World!\n";
    return 0;
}
```

Explanation

// indicates that everything following it until the end of the line is a comment. A comment is ignored by the compiler. Another way to write a multiple-line comment is to put it between /* and */
Comments exist to explain non-obvious things going on in the code.

#include: a preprocessor command to include the contents of another file, here the **iostream** file, which defines the procedures for input/output.

int main() {...} defines the code that should execute when the program starts up. The curly braces represent a grouping of multiple commands into a block.

cout << : This is the syntax for outputting some piece of text to the screen.

Explanation

Namespaces: In C++, identifiers can be defined within a context – sort of a directory of names – called a namespace. When we want to access an identifier defined in a namespace, we tell the compiler to look for it in that namespace using the scope resolution operator (::). Here, we're telling the compiler to look for cout in the std namespace, in which many standard C++ identifiers are defined. A better alternative is to add the following line below line 2:

using namespace std ;"

Strings: A sequence of characters such as "**Hello, world** " is known as a string.

return 0 indicates that the program should tell the operating system it has completed successfully.

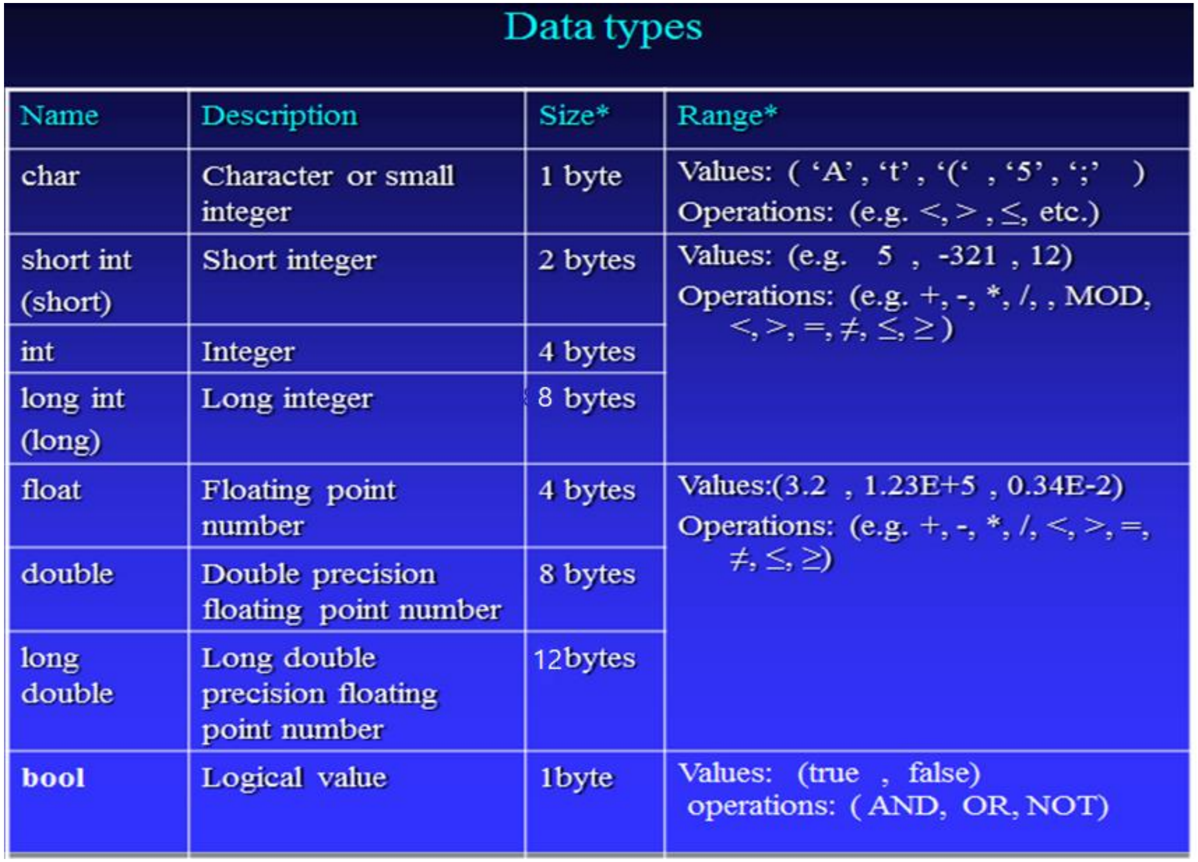
Basic definitions

- A statement is a unit of code that does something – a basic building block of a program. Example: *cout<< “C++ is fun”;*

- An expression is a statement that has a value – for instance, a number, a string, the sum of two numbers, etc. Example: *4 + 2, x - 1*

Not every statement is an expression. It makes no sense to talk about the value of an #include statement, for instance.

Data types



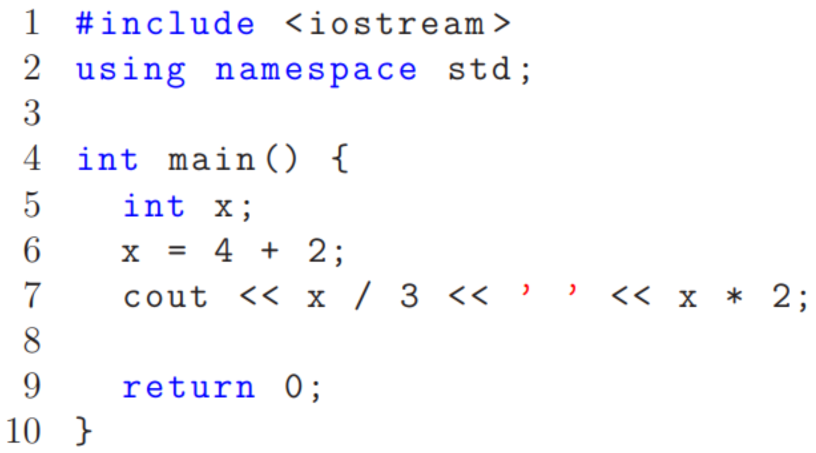
Data types

Name	Description	Size*	Range*
char	Character or small integer	1 byte	Values: (‘A’ , ‘t’ , ‘(‘ , ‘5’ , ‘;’) Operations: (e.g. <, > , ≤, etc.)
short int (short)	Short integer	2 bytes	Values: (e.g. 5 , -321 , 12) Operations: (e.g. +, -, *, /, , MOD, <, >, =, ≠, ≤, ≥)
int	Integer	4 bytes	
long int (long)	Long integer	8 bytes	
float	Floating point number	4 bytes	Values:(3.2 , 1.23E+5 , 0.34E-2) Operations: (e.g. +, -, *, /, <, >, =, ≠, ≤, ≥)
double	Double precision floating point number	8 bytes	
long double	Long double precision floating point number	12bytes	
bool	Logical value	1byte	Values: (true , false) operations: (AND, OR, NOT)

Variables

We might want to give a value a name so we can refer to it later. We do this using variables.

A variable is a named location in memory. For example, say we wanted to use the value 4 + 2 multiple times. We might call it x and use it as follows:



```cpp
#include <iostream>
using namespace std;

int main() {
  int x;
  x = 4 + 2;
  cout << x / 3 << ' ' << x * 2;

  return 0;
}
```

Variables

The name of a variable may contain numbers, letters, and underscores (_), BUT should not start with a number or contain a white space.

It should not also be a reserved word, for example: include, main, for, if .

- **Example of a valid variable name:**

area , length, X , Y1, abc, d3, st_number

- **Example of an invalid variable name:**

2Y { begins with a digit }

Ali's { contains the symbol ' }

st-age { the symbol - is not underscore }

while { it is a keyword }

ab cd { it has a space }

NOTE: C++ is a case sensitive language

Declaration and initialization

- Line 5 in the previous example which is **int x** is the **declaration** of the variable x.
- We must tell the compiler **what type** x will be so that it knows *how much memory to reserve for it and what kinds of operations may be performed on it*.
- Line 6, which is **x = 4+2,** is the **initialization** of x, where we specify an initial value for it. This introduces a new operator: =, the **assignment** operator. We can also change the value of x later on in the code using this operator.
- We could replace lines 5 and 6 with a single statement that does both declaration and initialization: **int x = 4 + 2**;

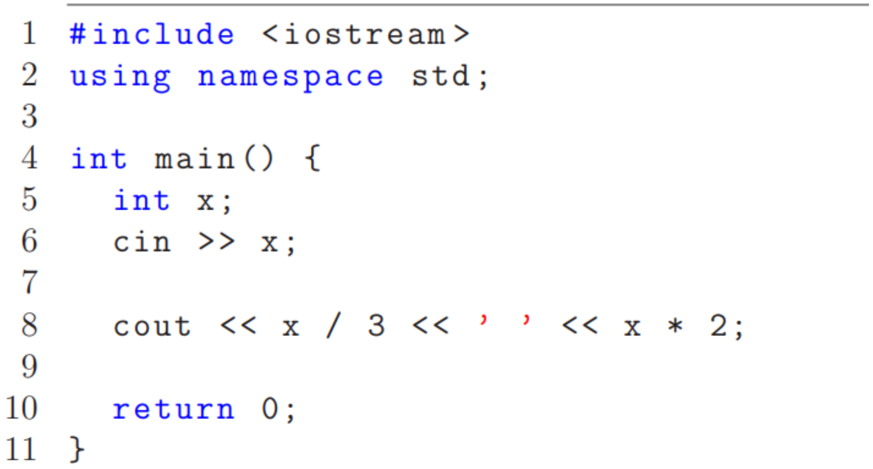
Declaration and initialization

Examples:

- int c=5;
- int n1=3, n2=7;
- float c1=3.5, c2;
- int z;
  z=7;
- char letter= ‘e’;
- string color = “red”;

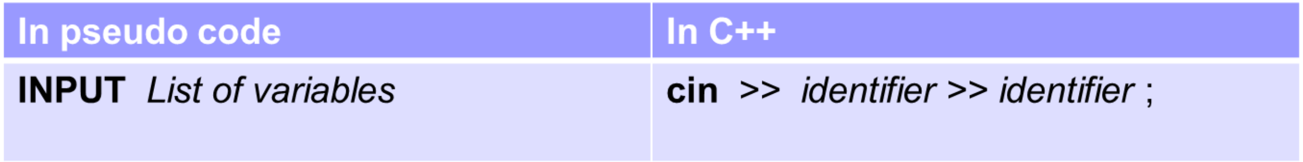
Input

Now that we know how to give names to values, we can have the user of the program input values. This is demonstrated in line 6 below:



```
 1  #include <iostream>
 2  using namespace std;
 3
 4  int main() {
 5    int x;
 6    cin >> x;
 7
 8    cout << x / 3 << ’ ’ << x * 2;
 9
10    return 0;
11  }
```

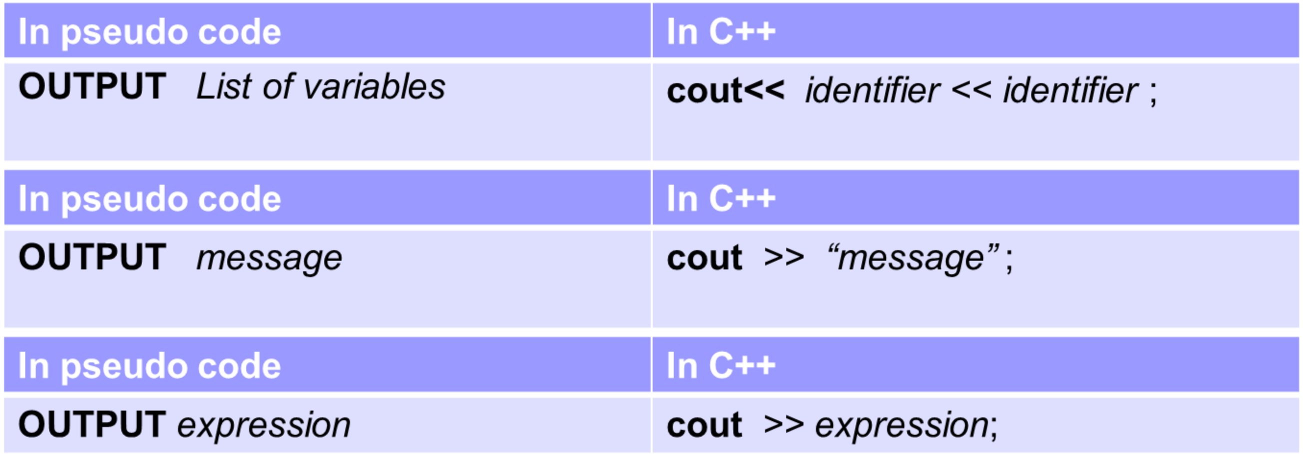
Input



In pseudo code	In C++
INPUT *List of variables*	**cin** >> *identifier* >> *identifier* ;

Example: cin>> length;

Output



In pseudo code	In C++
OUTPUT *List of variables*	**cout<<** *identifier* << *identifier* ;

In pseudo code	In C++
OUTPUT *message*	**cout** >> *“message”* ;

In pseudo code	In C++
OUTPUT *expression*	**cout** >> *expression*;

Example: cout<< “length=”<<length;

Assignment

Storing a new value in a memory location is called an **assignment**.

In pseudo code	In C++
Variable ← Expression	Variable = Expression

The semantics (execution) of this statement:

1- The Expression on the RHS is evaluated

2- The result of the expression is assigned to the variable on the LHS

Assignment

NOTE:

The right hand side (RHS) of the assignment statement should be of the same data type of the left hand side (LHS).

Examples:

1- T ← true

This will be correct if T is of Boolean type.

2- A ← x + y * 2

This will be correct if A has a numeric data type (e.g. integer, or real) and the value of the expression on (RHS) has the same numeric data type.

Assignment

L.H.S = R.H.S.

X+ 3 = y + 4 **Wrong**

Z = x +4 **True**

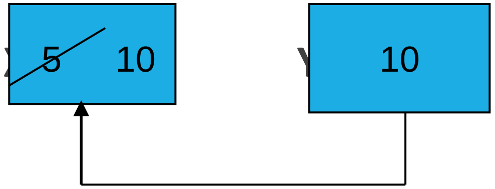
x +4 = Z **Wrong**

Assignment

If we want to copy a value from one memory location (say, X) into another location (say, Y), we say that we **dereference** a variable.

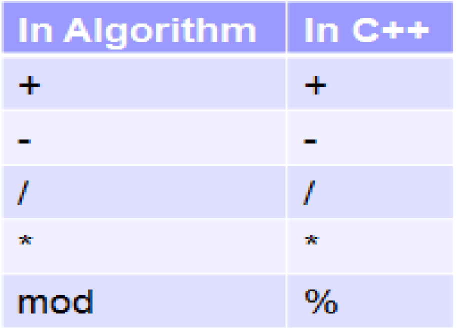
e.g.

```
X ← 5
Y ← 10
X ← Y        // now X has the value 10
```



Operators and Expressions

Arithmetic operators:



In Algorithm	In C++
+	+
-	-
/	/
*	*
mod	%

An Arithmetic expression is composed of operands and arithmetic operations.

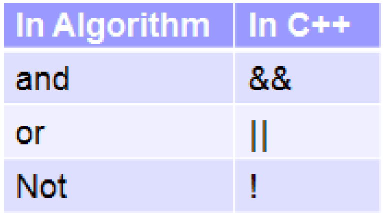
- Operands may be numbers and/or identifiers that have numeric values

- Its result is a numeric value

EX: T MOD 2 gives 0 if T is any even number, and 1 if T is any odd number

Operators and Expressions

Logical operators



In Algorithm	In C++
and	&&
or	||
Not	!

A **Logical Expression** is also called a **Boolean expression**.

- It is composed of operands that have logical values and logical operators.

- Its result is a logical value (**true** or **false**)

Operators and Expressions

The truth table

(1) **AND table**

AND	True	False
True	True	False
False	False	False

Operators and Expressions

(2) OR table

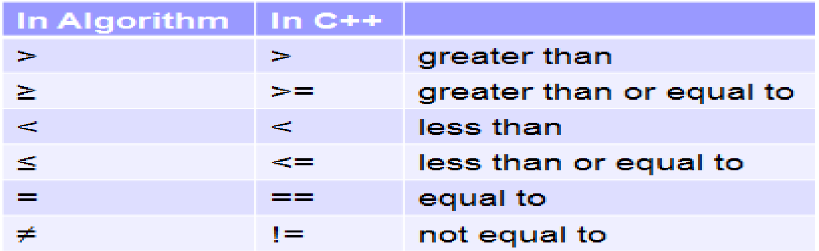
OR	True	False
True	True	True
False	True	False

(3) NOT table

NOT	True	False
	False	True

Operators and Expressions

Relational operators:



In Algorithm	In C++	
>	>	greater than
≥	>=	greater than or equal to
<	<	less than
≤	<=	less than or equal to
=	==	equal to
≠	!=	not equal to

- **A relation Expression** is composed of operands and operators.

  - Operands may be numbers and/or identifiers that have numeric values.

  - Its result is a logical value (**true** or **false**)

Operators and Expressions

NOTES

1) A relational expression may contain arithmetic sub-expressions,

e.g. (3 + 7) **<** (12 * 4)

2) A logical expression may contain relational and arithmetic sub-expressions,

e.g.

1- x **AND** y **AND** (a **>** b)

2- (2 + t) **<** (6 * w) **AND** (p **=** q)

Operator Precedence

Expressions are evaluated according to the precedence rule.

Precedence Rule:

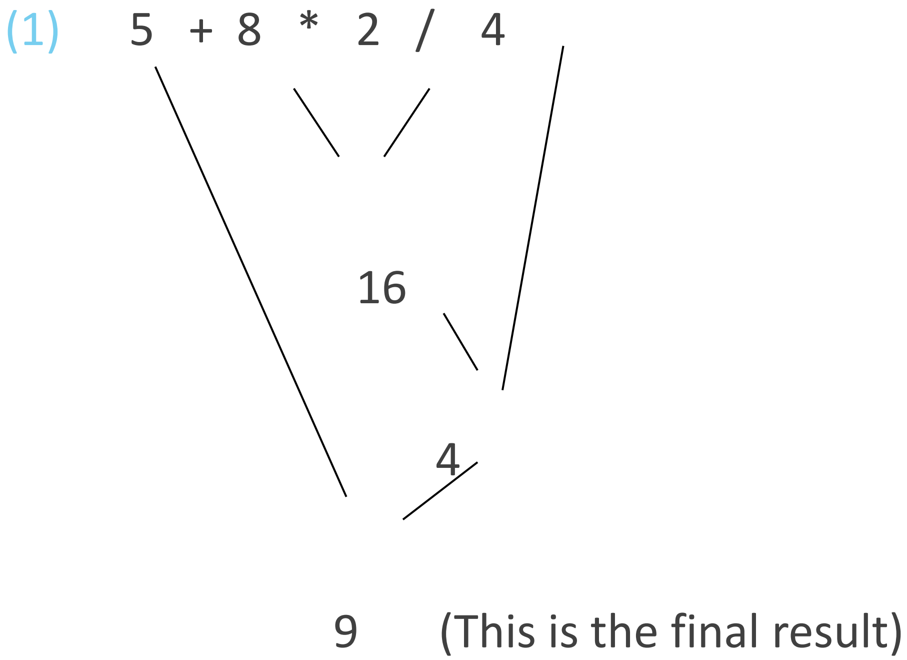
- Each operator has its own precedence that indicates the order of evaluation.
- If the expression has operators of the same precedence, then the evaluation starts from the left of the expression to the right.

Operator Precedence

Operator In pseudo code	**Operator In** C++	**Description**	**Precedence**
(	(	parentheses	Higher
not	!		
*, /, MOD	*, /, %		
+ , -	+ , -	Binary plus, binary minus	
<, , ≤, >, ≥	<, <=, >, >=		
= , ≠	== , !=	Equal, not equal	
AND	&&		
OR	||		
←	=	Assignment	Lower

Examples

Find the value of the following expression:



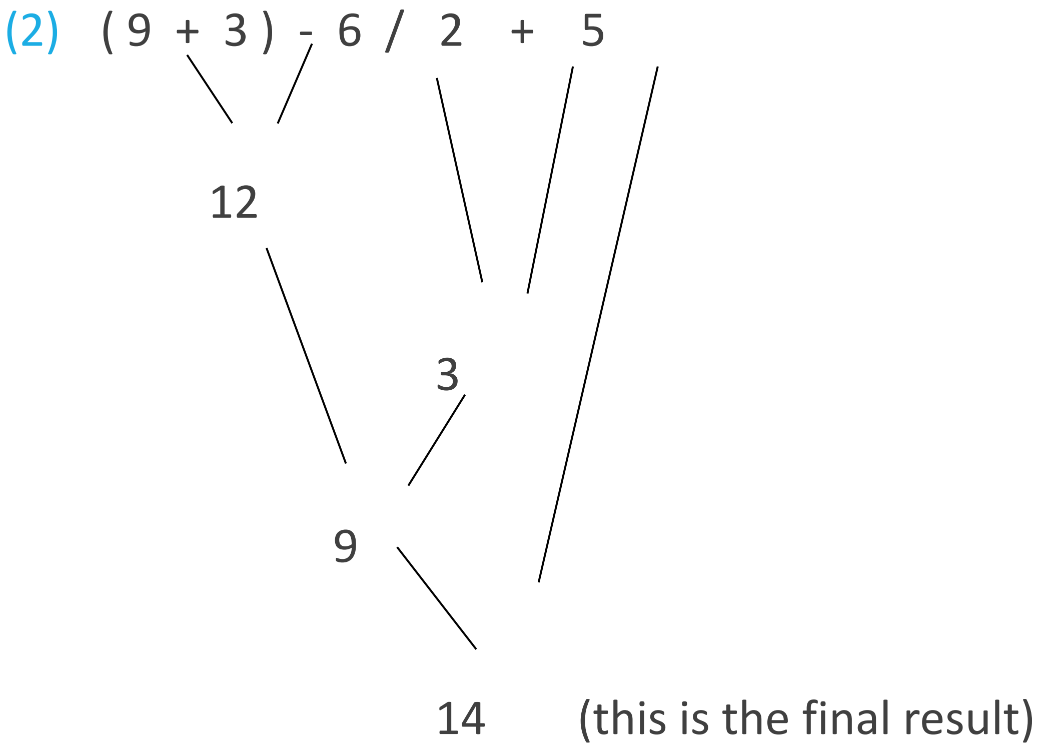
(1) 5 + 8 * 2 / 4

16

4

9 (This is the final result)

Examples .. cont.



(2) (9 + 3) - 6 / 2 + 5

12

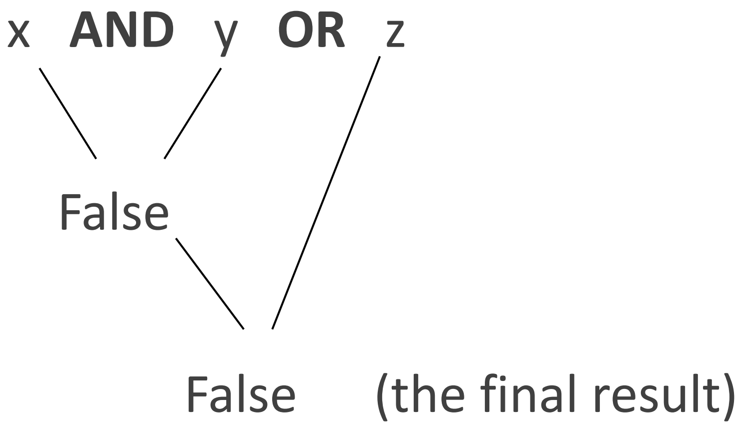
3

9

14 (this is the final result)

Examples on Logical Expressions

(1) If x = True, y = False, z = False, find the value of the expression x **AND** y **OR** z



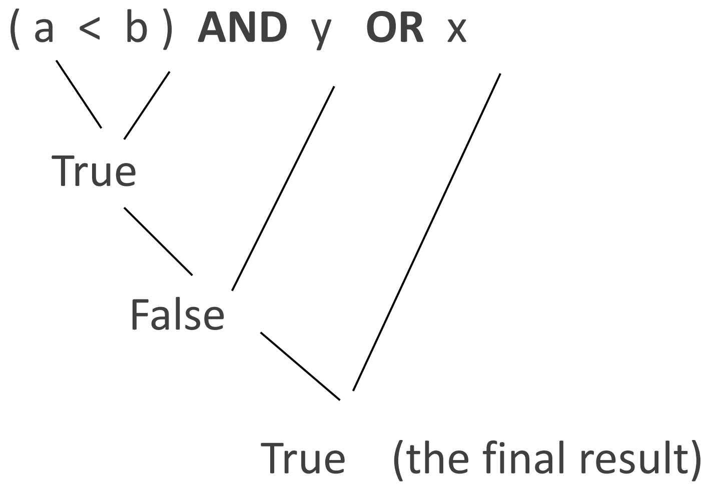
x **AND** y **OR** z

False

False (the final result)

Examples on Logical Expressions .. cont.

(2) If a = 3, b = 5, x = true, y = false, find the value of the expression: (a < b) AND y OR x



(a < b) **AND** y **OR** x

True

False

True (the final result)

Short circuiting:

- Short circuiting means that we don't evaluate the second part of an AND or OR unless we really need to.