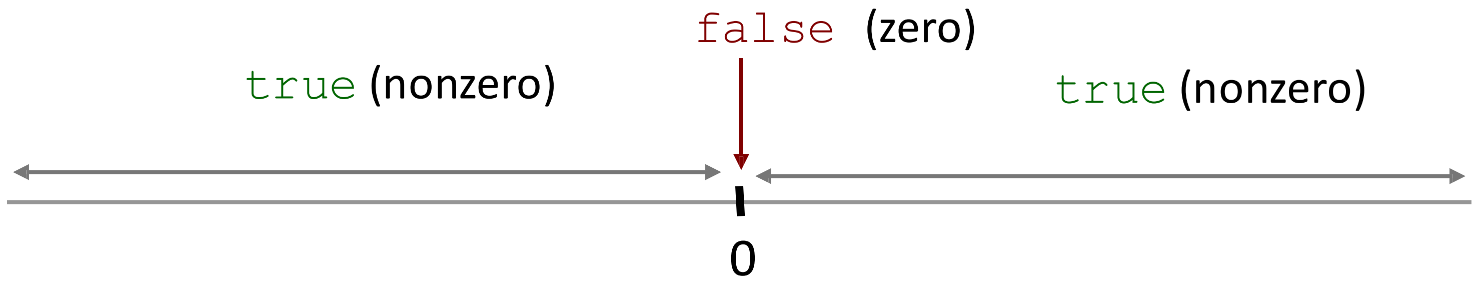
Selection Structures: Supplement

UNIVERSITY OF WOLLONGONG IN DUBAI

What is `true` and what is `false` in C

- C uses integer values to represent boolean `true` and `false`



```c
#include <stdbool.h>
………
int flag, p=100, q=50;
bool isBigger = false;

flag = (p <= q);
isBigger = (p >= q);
```

```
flag = 0;          false
isBigger = 1;      true
```

- You need to include `<stdbool.h>` to use `bool` data type in your program

Relational Operators

Purpose:

Compare two operands

Syntax:

`Operand1` **Relational Operator** `Operand2`

Operands:

constants

variables

arithmetic expressions

function calls

Operators

==	equal
!=	not equal
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Examples:

```
y >= 20
offset == (640 + x)
z < log(y)
```

Logical operators

Syntax:

```
Operand1 LogicOperator Operand2
```

or `!Operand`

Examples:

```
(current >= 2.0e-3) && (current <= 5.0e-3)
ans = (p > 95) || (q < 95);
ans = !(p > 95);
```

Operands:
- `bool` constants
- `bool` variables
- relational expressions
- function calls

Operators:

`&&`	logical AND
`||`	logical OR
`!`	logical NOT

Expression result:
- `true`
- `false`

Apart from difference in syntax, all properties of C logical operators are similar to those used for digital hardware design (covered by ECTE233) and in MATLAB (covered by ENGG100)

Logical operators

Evaluation of logical expressions

x	y	x && y	x || y	!x
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Truth table

As follows from the table:

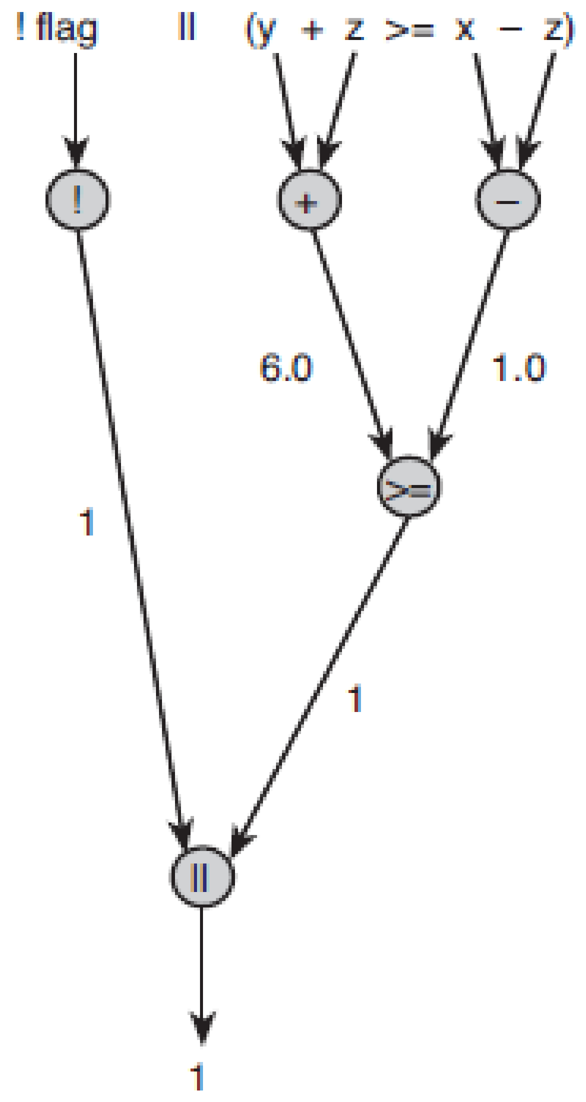
- If at least one operand is `false`, then `&&` is also `false`
- If at leas one operand is `true`, then `||` is also `true`

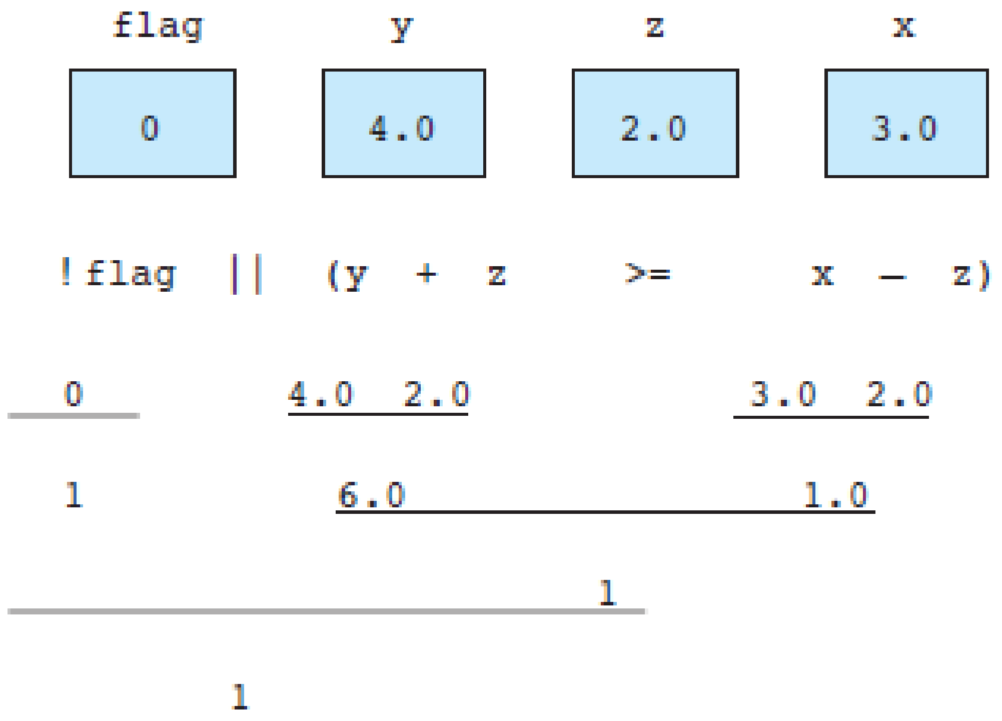
```
int x=2, y=4, z=9, flag;
bool isMaxVal;

            true       true
isMaxVal = (z > x) && (z > y);   /* true */
        false        true
flag = (x == y) && ( x < y );    /* false */
```

Operator Precedence

Operator	Precedence
function calls	highest (evaluated first)
! + - & (unary operator)	↓
* / %	↓
+ -	↓
< <= >= >	↓
== !=	↓
&&	↓
||	↓
=	lowest (evaluated last)

Evaluation Tree and Step-by-Step Evaluation for !flag || (y + z >= x - z)





Comparing Characters

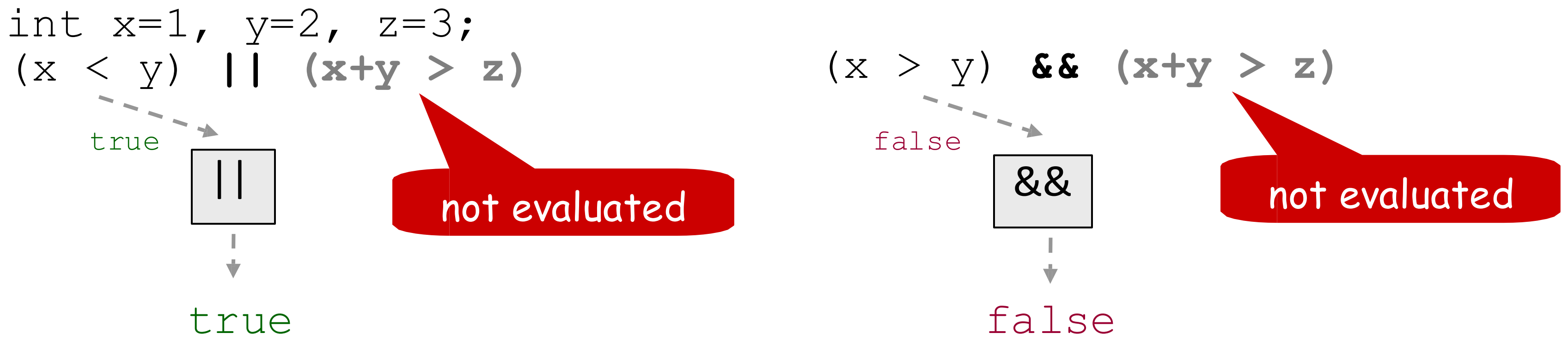
Expression	Value
‘9’ >= ‘0’	1 (true)
‘a’ < ‘e’	1 (true)
‘B’ <= ‘A’	0 (false)
‘Z’ == ‘z’	0 (false)
‘a’ <= ch && ch <= ‘z’	1 (true) if ch is a lowercase letter

Short circuit evaluation

Short-Circuit evaluation

- C stops further evaluation of an expression when the result becomes obvious (MATLAB uses short-circuit evaluation too)
  - –An expression of the form **`A||B`** **is always `true` if `A` is `true`**
  - –An expression of the form **`A&&B`** **is always `false` if `A` is `false`**

In these cases B is not evaluated



```
int x=1, y=2, z=3;
(x < y) || (x+y > z)
```

```
(x > y) && (x+y > z)
```

Short circuit evaluation

Short circuit evaluation reduces computational complexity of complex logical expressions

Does it have any side effects?

```
int x=1, y=2, z=3;
bool flag;

flag = x<0 || ++y > 7;        /* flag= false   y= 3 */
flag = x>0 || ++y > 7;        /* flag= true    y= 2 */
```

Not run in sequence

- When you use complex logical expression in your program, make sure that short circuit evaluation does not introduce intermittent mistakes in calculations

UNIVERSITY OF WOLLONGONG IN DUBAI

Writing English Conditions in C

Table below shows some English conditions and the corresponding C expressions. Each expression is evaluated assuming `x` is `3.0`, `y` is `4.0`, and `z` is `2.0`.

English Condition	Logical Expression	Evaluation
`x` and `y` are greater than `z`	`x > z && y > z`	`1 && 1` is `1` (true)
`x` is equal to `1.0` or `3.0`	`x == 1.0 || x == 3.0`	`0 || 1` is `1` (true)
`x` is in the range `z` to `y`, inclusive	`z <= x && x <= y`	`1 && 1` is `1` (true)

`x` is outside the range `z` to `y`	`!(z <= x && x <= y)`
	`z > x || x > y`
		`!(1 && 1)` is `0` (false)
		`0 || 0` is `0` (false)

UNIVERSITY OF WOLLONGONG IN DUBAI

Quiz

What are the values of the variables after execution of this code?

```
int a=0, b=0;

if (a > b || a==0 && b>0)
{
   a++;
   b += 2;
}
else
{
   a--;
   b -= 2;
}
```

```
int a=0, b=0;

if (a == 0)
   if (b > 0)
   {
      a++;
      b += 2;
   }
   else if (a > b)
   {
      a++;
      b += 2;
   }
   else
   {
       a--;
       b -= 2;
   }
```

a = -1 b = -2

a = -1 b = -2

Typical bugs

- Unnecessary semicolon after the `if(condition)`

```
if(number > THRESHOLD);   Null statement, do nothing
   number = 0;  <-  will be executed unconditionally
```

- Missing braces `{…}` to enclose compound statements

```
 if(number > THRESHOLD)
       number = 0;
else
       number = getNewNumber;
       counter++;  <-will be executed unconditionally
```

- Dangling `else`

```
if(a > 0)
     if(b < 10)
         c = 5;
else                          <-  else is always associated with the previous if
   c = 0;
```