

The slide features a decorative arrangement of five circles. Two are solid purple, and three are hollow with a purple outline. They are positioned around the text.

Informatics

Ingeniería en Electrónica y Automática Industrial

Operators and expressions in C

V1.1

© Autores

The slide features a decorative arrangement of five circles. Two are solid purple, and three are hollow with a purple outline. They are positioned around the text.

Operators and expressions in C

- Numerical expressions and operators
- Arithmetical operators
- Relational and logical operators
- Bitwise operators
- Assignment operators and expressions
- Other operators
 - Conditional operator
 - Comma operator
 - Address and indirection operators
 - `sizeof` operator
- Precedence and order of evaluation
- Type conversions

V1.1

© Autores

2

Numerical expressions and operators

- A **numerical expression** is formed by
 - Operators
 - Operands
- An **operator** is a symbol that indicates how the operands must be processed in the expressions
- An **operand** is the object that is processed: variables, constants, etc.

V1.1

© Autores

3

Arithmetical operators

- If the operands are of different types, the lower precision ones are transformed to the greater type

OPERATOR	OPERATION	OPERANDS
+	Addition	Integers or reals
-	Subtraction	Integers or reals
*	Multiplication	Integers or reals
/	Division	Integers or reals
%	Modulus: Remainder of integer division	Integers
-	Unary minus (sign change)	Just one operand (integer or real)

V1.1

© Autores

4

Relational and logical operators (I)

- Operands can be of any type but the result is always an integer with just two possible values: 1 (*true*) or 0 (*false*)

LOGICAL OPERATORS	
OPERATOR	OPERATION AND RESULT
&&	Logical AND . Result is 1 if both operands are non-zero (ie. if one or both are 0, result is 0).
	Logical OR . Result is 1 if any of the operands is non-zero (ie. result is 0 just when both operands are 0).
!	Logical NOT . Result is 1 if the operand is 0, and 0 otherwise.

V1.1

© Autores

5

Relational and logical operators (II)

RELATIONAL OPERATORS	
OPERATOR	OPERATION AND RESULT
<	Result is 1 if the left operand is lower than the right one; 0 otherwise.
>	Result is 1 if the left operand is greater than the right one; 0 otherwise.
<=	Result is 1 if the left operand is lower than or equal to the right one; 0 otherwise.
>=	Result is 1 if the left operand is greater than or equal to the right one; 0 otherwise.
!=	Result is 1 if the operands are different ; 0 otherwise.
==	Result is 1 if the operands are equal ; 0 otherwise.

V1.1

© Autores

6

Bitwise operators

- They operate with the individual bits of the operands, which must be integer type (`int` or `char`)

OP.	OPERATION AND RESULT
<code>&</code>	AND between bits of the operands
<code> </code>	OR between bits of the operands
<code>^</code>	XOR (Exclusive OR) between bits of the operands
<code>~</code>	1'Complement of the operand (at the right of the operator)
<code><<</code>	Left shift of the left operand by the number of positions given by the positive right operand (filling vacants with zeros).
<code>>></code>	Right shift of the left operand by the number of position given by the positive right operand. If the operand is <code>unsigned</code> fills vacants with zeros, if signed, fill vacants with sign bit (arithmetic shift).

V1.1

© Autores

7

Assignment operators (I)

- They assign values to one variable
 - Simple assignment operator =
 - `a = 2`
 - Increment `++` and decrement `--` operators
 - Increment/decrement by 1 the value of a variable in a expression
- `++variable`. First increment, later use.
- ```
x = 1;
y = ++x; // x is now 2, y is also 2
y = x++; // x is now 3, y is 2
```
- `variable++`. First use, later increment
- ```
x = 3;
y = x--; // x is now 2, y is 3
y = --x; // x is now 1, y is also 1.
```

V1.1

© Autores

8

Assignment operators (II)

- Operation *and* assignment

`variable (op)= expression;`

Is equivalent to

`variable = variable (op) expression;`

- `(op)` is the assignment operator
- `expression` is the expression that will be evaluated along with `variable` to obtain its new value

`i += 2`

`i = i + 2`

are equivalent expressions

V1.1

© Autores

9

Assignment operators (III)

OP.	OPERACIÓN Y RESULTADO
<code>*=</code>	Multiplication and assignment. <code>C *= A</code> equiv. to <code>C = C*A</code>
<code>/=</code>	Division and assignment. <code>C /= A</code> equiv. to <code>C = C/A</code>
<code>%=</code>	Modulus and assignment. <code>C %= A</code> equiv. to <code>C = C%A</code>
<code>+=</code>	Addition and assignment. <code>C += A</code> equiv. to <code>C = C+A</code>
<code>-=</code>	Subtraction and assignment. <code>C -= A</code> equiv. to <code>C = C-A</code>

V1.1

© Autores

10

Assignment operators (IV)

OP.	OPERATION (bit level) AND RESULT
<code><<=</code>	Left shift AND assignment <code>C <<= 2</code> equiv. to <code>C = C<<2</code>
<code>>>=</code>	Right shift AND assignment <code>C >>= 2</code> equiv. to <code>C = C>>2</code>
<code>&=</code>	Bitwise AND and assignment <code>C &= 2</code> equiv. to <code>C = C&2</code>
<code> =</code>	Bitwise OR and assignment <code>C = 2</code> equiv. to <code>C = C 2</code>
<code>^=</code>	Bitwise XOR and assignment. <code>C ^= 2</code> equiv. to <code>C = C^2</code>

V1.1 © Autores 11

Other operators (I)

- **Condicional Operator «?:»**

`expr1 ? expr2 : expr3`

- If `expr1` is true, then `expr2`, is evaluated
- If `expr1` is false, then `expr3` is evaluated
- Ex. `(a >= b) ? puts("a>=b") : puts("b>a");`

- **Comma Operator « , »**

- Mostly used in the `for` statement.
- When used to concatenates expressions and variables or to separates elements in argument lists is NOT an operator (do not guarantee left to right evaluation)

V1.1

© Autores

12

Other operators (II)

- **Address operator «&»**
 - `&variable` obtains the memory address of `variable`
- **Indirection operator «*»**
 - `*identifier` refers to the content of memory address `identifier`
- **Operator «sizeof»**
 - Returns the number of bytes that the operand occupies in memory

V1.1

© Autores

13

Precedence and order of evaluation (I)

Order	OPERATORS											ASOCIATIVITY	
1º	()	[]	.	->	sizeof								Left to Right
2º	-	~	!	*	++	--	(tipo)						Right to Left
3º	*	/	%										Left to Right
4º	+	-											Left to Right
5º	<<	>>											Left to Right
6º	<	<=	>	>=									Left to Right
7º	==	!=											Left to Right
8º	&												Left to Right
9º	^												Left to Right
10º													Left to Right
11º	&&												Left to Right
12º													Left to Right
13º	?:												Right to Left
14º	=	*=	/=	%=	+=	-=	<<=	>>=	&=	=	^=		Right to Left
15º	,												Left to Right

V1.1

© Autores

14

Precedence and order of evaluation (II)

- Precedence and order of evaluation (table)
 - Operators in the same line have the same priority
 - Priority decreases from top to bottom
 - Parenthesis are evaluated from inside to outside (as usual)
 - Some ambiguities may exist depending on the compiler

USE PARENTHESIS!! when doubting

V1.1

© Autores

15

Type conversions (I)

- In expressions operands are type-converted automatically
 - With reals involved, all are converted to the high precision one
 - Real constants are `double` by default
 - `char` and `short` are converted to `int` or to `unsigned int`
 - With integers involved, all are converted to the longest one

EXAMPLE

```
long a
char b;
int c, f;
float d;
f = a + b*c/d ;
```

- `b` is converted to `c` type (`int`) and `b*c` is `int`
- `b*c` is converted to `float` and divided by `d`
- `a` is converted to `float` and added to `b*c/d`.
- `float a+b*c/d` is converted to `int` (eliminating fractional part) and saved in integer `f`

Better try not to mix types...

V1.1

© Autores

16

Type conversions (III)

- **Explicit conversion: «(cast)» operator**

```
(newtype)expression;
```

- Example:

- `7/2` gives 3 as result
- `(float)7/2` gives 3.5 as result