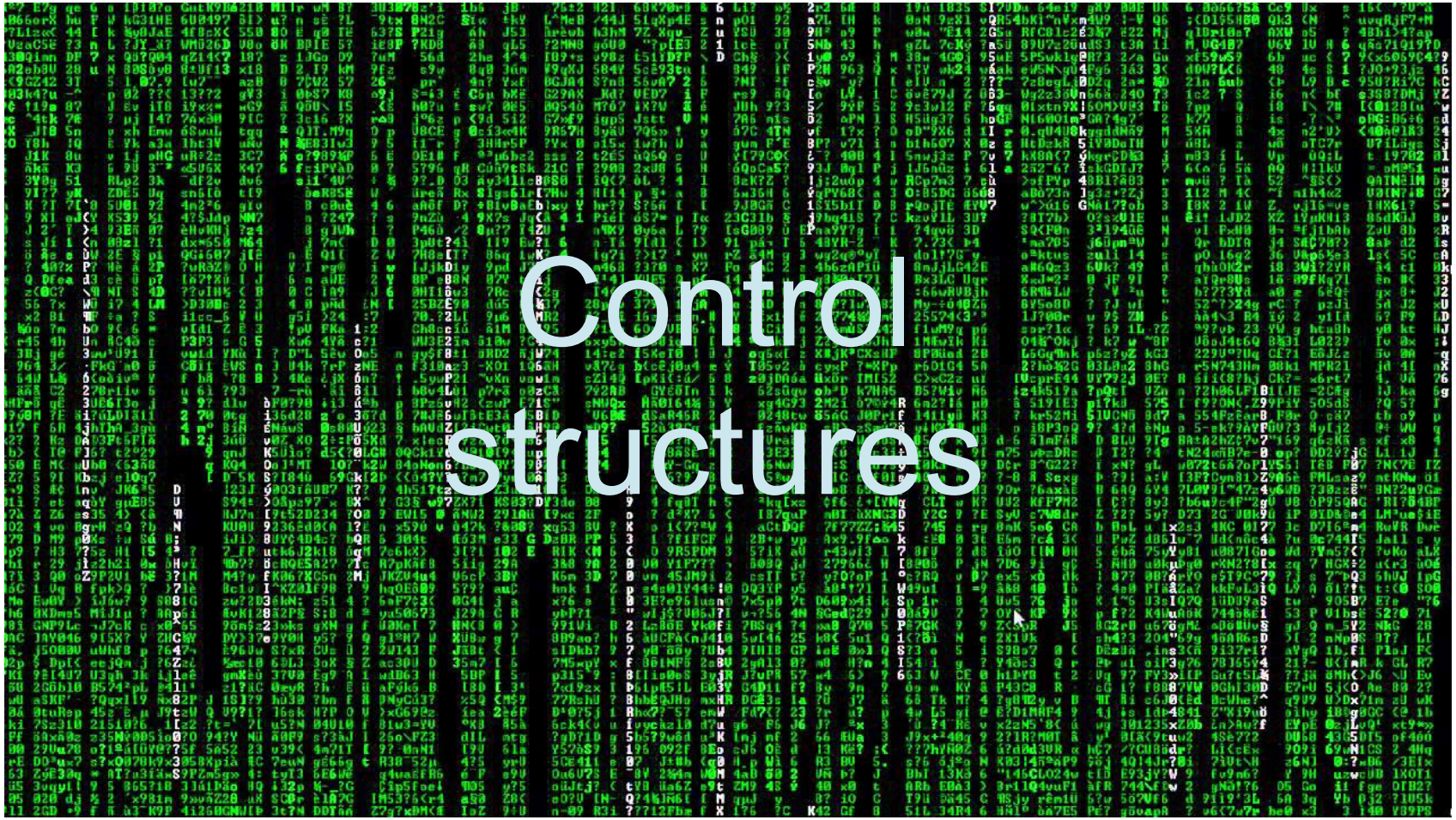
ESIGELEC
ÉCOLE D'INGÉNIEURS-ES GÉNÉRALISTES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# C Programming




Ángel Perles
angel.perles@upv.es
R02

# Contents

Objectives
Conditions
Braces and if … else chains
Switch
Loops
Activities and summary

# Objectives

- Use if, switch, while, do-while and for in C
- Explain how C evaluates a condition (0 / non-zero)
- Spot code that Java rejects and C accepts
- Apply the BARR-C layout: braces and switch
- Avoid infinite loops with unsigned counters
- Exit nested loops without labeled break

# You already know this

| Construct | Java | C |
|---|---|---|
| `if / else`, `? :` | ✓ | ✓ same syntax |
| `while`, `do … while`, `for` | ✓ | ✓ same syntax |
| `break`, `continue` | ✓ (also labeled) | ✓ unlabeled only |
| `switch` / `case` / `default` | ✓ (also String, ->) | ✓ integers only |
| `&&`, `\|\|`, `!` (short-circuit) | ✓ | ✓ same |
| for-each `for (x : arr)` | ✓ | ✗ does not exist |

Same syntax. What changes: conditions, switch, loops with unsigned and the style.

# CONDITIONS

```
if
```

# A condition is just an integer

- `0` → false; **any other value** → true
- `==`, `<`, `&&`, `!`… give an `int`: 0 or 1
- `bool`, `true`, `false`: `<stdbool.h>` (C99), still 0 / 1
- Java needs a `boolean`; C takes any integer or pointer

JAVA

```
if (count) { }     /* error */
if (count != 0) { }
```

C

```
if (count)              /* compiles */
if (strcmp(s1, s2))     /* true if DIFFERENT! */
if (0 == strcmp(s1, s2))  /* equal */
```

# Java rejects it, C compiles it

| Code | What C does | gcc -Wall -Wextra |
|---|---|---|
| `if (x = 5)` | Assigns 5, then tests 5 → always true | *suggest parentheses around assignment used as truth value* |
| `if (0 < x < 10)` | (0 < x) is 0 or 1, always < 10 → x = 20 is "in range" | *comparisons like 'X<=Y<=Z' do not have their mathematical meaning* |
| `if (x & MASK == 0)` | == binds tighter than & → x & (MASK == 0) | *suggest parentheses around comparison in operand of '&'* |

Fixes: **`if (5 == x)`** · **`if ((0 < x) && (x < 10))`** · **`if (0 == (x & MASK))`**

# BARR-C: writing conditions

- Constant on the **left** of `==`: a typo `5 = x` becomes an **error**
- Parentheses around **every** sub-expression; never rely on precedence
- No assignments inside `if` / `else if` tests
- Explicit comparisons: `0 == n`, `NULL == p`

```
if (NULL == p_state)
{
    return;
}

if ((raw > SENSOR_ADC_MAX) ||
    (SENSOR_FAULT_VALUE == raw))
{
    sensor_ok = false;
}
```

# BARR-C: braces, always

JAVA HABIT (also C habit)

```
if (level > max) {
    alarm();
} else
    pump_on();
```

BARR-C

```
if (level > max)
{
    alarm();
}
else
{
    pump_on();
}
```

- Braces **always**, even for one statement
- `{` alone on the next line; `}` aligned with it
- 4 spaces per level, no tabs
- One space after `if`, `while`, `for`, `switch`

# Why always braces: goto fail

```
if ((err = SSLHashSHA1.update(&hashCtx,
                 &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx,
                 &hashOut)) != 0)
    goto fail;
...
fail:
    return err;    /* err == 0: "OK" */
```

- Apple TLS code (2014), CVE-2014-1266
- The 2nd `goto` is **not** inside the `if`: it always runs
- Signature check skipped, `err` is 0 → certificate accepted
- With braces the bug is visible
- Today gcc `-Wall` warns: *this 'if' clause does not guard…*

# if … else if chains

```
if (fill_pct < p_config->low_pct)
{
    status = TANK_LOW;
}
else if (fill_pct > p_config->high_pct)
{
    status = TANK_HIGH;
}
else
{
    status = TANK_NORMAL;
}
```

- BARR-C: a chain with `else if` always ends with `else`
- Nesting: at most 2 levels; deeper → a function
- No `elsif` / `elif`: `else if` is just `else` + `if`

# SWITCH



# switch: BARR-C layout

```
switch (status)
{
    case TANK_LOW:
        printf("LOW");
    break;

    case TANK_HIGH:
        printf("HIGH");
    break;

    default:
        printf("OTHER");
    break;
}
```

- `break` aligned with its `case`, not with the body
- Every `case` ends with `break`
- Always a `default`, last, even with an enum
- Blank line between cases
- Intentional fall-through: say it in a comment `/* Fall through - also reset. */`

# switch: differences with Java

| | Java | C |
|---|---|---|
| Selector | int, char, `String`, enum | **Integer types only** (char, int, enum…); strings → `if` + `strcmp()` |
| `case` label | constant, `final` allowed | `const` variable → **error**; use `#define` or an `enum` |
| enum | type-safe object | just an int: any value can arrive → `default` |
| Fall-through | same | same; `-Wextra` warns *this statement may fall through* |
| New forms | `case X ->`, switch expressions | do not exist |

# LOOPS

# for

# Loops: the C details

- Counter declared in the `for`: needs C99
- No for-each: arrays have no `.length` → index + size
- Infinite loop (BARR-C): `for (;;)`
- Empty body: braces + comment
- No magic numbers in the loop test

EMBEDDED "SUPERLOOP"

```
int main(void)
{
    hw_init();

    for (;;)
    {
        read_inputs();
        update_outputs();
    }
}
```

# Unsigned loops that never end

NEVER ENDS

```
/* Count down */
for (uint8_t i = n - 1U; i >= 0U; i--)

/* Count to 300 */
for (uint8_t i = 0U; i < 300U; i++)
```

gcc **-Wextra**: *comparison is always true due to limited range of data type*

FIX: COUNT DOWN

```
for (uint8_t i = n; i > 0U; i--)
{
    printf("%u\n", i - 1U);
}
```

Java has no unsigned types: this trap is new. Size the counter for the range.

# No labeled break in C

JAVA

```
outer:
for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
        if (grid[i][j] == target) {
            break outer;
        }
    }
}
```

C: A FLAG IN THE CONDITION

```
bool found = false;

for (uint8_t i = 0U; (i < NUM_ROWS) && !found; i++)
{
    for (uint8_t j = 0U; (j < NUM_COLS) && !found; j++)
    {
        if (grid[i][j] == target)
        {
            found = true;
        }
    }
}
```

Or: move the loops to a function and **`return`**. **`goto`** exists, but BARR-C says avoid it.

# Activity: what does it print?

A

```
int32_t x = 20;

if (0 < x < 10)
{
    printf("in\n");
}
else
{
    printf("out\n");
}
```

B (code = 1)

```
uint8_t n = 0U;

switch (code)
{
    case 1:
        n++;
    case 2:
        n++;
    break;

    default:
        n = 10U;
    break;
}
printf("%u\n", n);
```

C

```
if (strcmp("abc", "abc"))
{
    printf("equal\n");
}
```

Then: which warnings does gcc give for A, B and C?

# Activity: fix and restyle

```
if(level>high) status=TANK_HIGH; else if(level<low) status=TANK_LOW;
switch(status){ case TANK_HIGH: valve_close(); case TANK_LOW: valve_open(); break; }
```

- Rewrite it with the BARR-C layout (braces, `break`, `default`)
- Find the **bug** that the layout makes visible
- Which case is missing in the `if` chain?

# Summary: Java → C

| Topic | Remember in C |
|---|---|
| Conditions | 0 = false, non-zero = true; explicit comparisons, constant on the left |
| Traps | `=` vs `==`, `a < x < b`, `&` vs `==` precedence, `strcmp()` returns 0 if equal |
| Braces | Always, on their own line (goto fail) |
| switch | Integers only; `break` aligned with `case`; always `default` |
| Loops | `for (;;)`; no for-each; watch unsigned counters |
| break | No labels: flag or function + `return` |

A warning is fixed like an error: gcc finds most of these for you.

ESIGELEC

ÉCOLE D'INGÉNIEURS-ES GÉNÉRALISTES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA