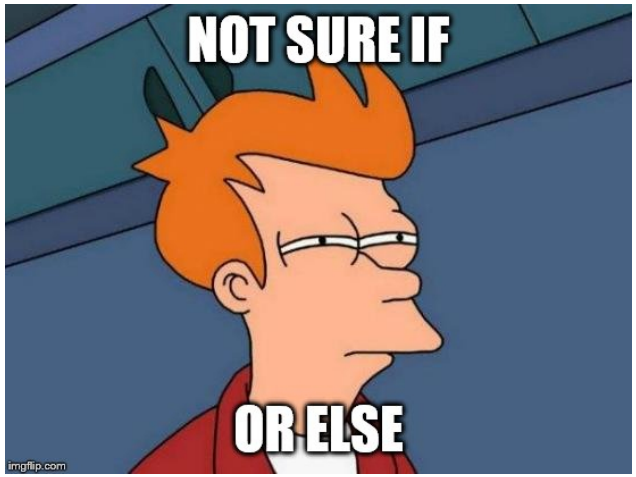
Conditional Execution

Pedro Ribeiro

DCC/FCUP

2024/2025



(based and/or partially inspired by Pedro Vasconcelos's slides for Imperative Programming)

Relational operators

- Binary operators for comparisons between numerical expressions:

`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to
`==`	equal
`!=`	different

- The result of a comparison is an integer:
  - `0` if the condition is false; `1` if the condition is true.
- Example:

```
int i, j;
i = 2;
j = 4;
printf("%d\n", i > j);        // print 0
printf("%d\n", i < j);        // print 1
printf("%d\n", (i*i) == j);   // print 1
printf("%d\n", (i*j) >= 10); // print 0
```

Precedence and associativity

- Relational operators have lower precedence than arithmetic ones
  - `i + j < k - 1` is equivalent to `(i+j) < (k-1)`
- Relational operators *associate on the left*
- Note that the expression `i < j < k` is valid but does not test whether `j` is between `i` and `k` *(as it would happen in Python)*:
  - `<` associates to the left, so the expression is the same as `(i<j) < k`
  - i.e. it compares `k` with the result of the `i<j` comparison (`0` or `1`).
- The correct expression uses the conjunction of two conditions:
  - `i < j && j < k`

Equality comparisons

- Two operators: `==` (equal) and `!=` (different)
- Result is `0` (false) or `1` (true)

```
int i, j;
i = 2;
j = 3;
printf("%d\n", i == j);    // print 0
printf("%d\n", i+1 == j); // print 1
printf("%d\n", i != j);    // print 1
```

- Do not confuse assignments with comparisons:
  - `i = j` modifies the left-hand side; the result is the assigned value
  - `i == j` compares left and right sides; result is `0` or `1`

If statement

- The `if` statement conditionally executes a statement according to the result of an expression

- Simplest form:

  `if ( expression ) statement`

  Calculates the value of the expression; if the result is non-zero, then executes the statement

- In any case, it continues by executing the following instructions

```
if ( line_num == MAX ) line_num = 0;
// program continues
...
```

If statement

- If you want to conditionally execute more than one instruction, group them together in a block:

  ```
  { instructions }
  ```

- Curly braces force the compiler to treat a block of instructions as one
- Each instruction within the block ends with a semicolon
- We don't add semicolons after closing curly braces
- We can put it on a single line:

```
if (line_num == MAX) { line_num = 0; page_num ++; }
```

But it's clearer if we start with several lines:

```
if (line_num == MAX) {
    line_num = 0;
    page_num ++;
}
```

If ... else

- The `if` statement can include an else alternative:

```
if ( expression ) else statement
```

- The statement following the `else` is executed if the expression has a value of zero (i.e. is *false*)

- Example:

```
if (i > j) max = i;
else max = j;
```

If ... else

- You can include `if` statements inside other `if`s
- In this case it is useful to use **indentation** and curly braces to make the structure explicit and comprehensible:

```
if (i > j) {
    if (i > k)
        max = i;
    else
        max = k;
} else {
    if (j > k)
        max = j;
    else
        max = k;
}
```

If ... else

- There are those who prefer to always use curly braces:

```
if (i > j) {
    if (i > k) {
        max = i;
    } else {
        max = k;
    }
} else {
    if (j > k) {
        max = j;
    } else {
        max = k;
    }
}
```

- By always placing curly braces:
  - it's easier to add statements inside if or else (we avoid errors resulting from forgetting the curly braces when adding more than one statement)

Cascading ifs

- To test conditions in sequence we write multiple cascading `if` statements.

  For example:

```c
if (n < 0)
   printf("n negative\n");
else if (n == 0)
   printf("n zero\n");
else
   printf("n positive\n");
```

- Note how we lined up all else statements, to provide better comprehension

Example: calculating commissions

- When a broker sells shares, he charges a commission, the amount of which depends on the amount traded

- Let's write a program to calculate the commission according to the following table:

Amount	**Commission**
Up to €2500	€30 + 1.7%
€2500 - €7500	€55 + 0.66%
€7500 - €20K	€80 + 0.34%
€20K - €50K	€110 + 0.22%
Above 50K	€150 + 0.11%

- The minimum commission to be charged should be €40

Example: calculating commissions

- The `broker.c` (source code) program reads the value of the transaction, calculates the commission and prints it out:

```
Enter amount: EUR 30000
Commission: EUR 176.00
```

- The core of the program is a cascading sequence of `if`s to determine what range the value is in
- At the end we include an extra condition to ensure that we always charge the minimum amount

```c
#include <stdio.h>

int main(void) {
  float amount, commission;

  printf("Enter value: EUR ");
  scanf("%f", &value);

  if(value < 2500)
    commission = 30.0 + 0.017 * value;
  else if(value < 7500)
    commission = 55.0 + 0.0066 * value;
  else if(value < 20000)
    commission = 80.0 + 0.0034 * value;
  else if(value < 50000)
    commission = 110.0 + 0.0022 * value;
  else
    commission = 150.0 + 0.0011 * value;

  if (commission < 40.0)
    commission = 40.0;

  printf("Commission: EUR %.2f\n", commission);

  return 0 ;
}
```

Logical operators

- We can build complex conditions from simpler ones using logical operators:

`&&`	conjunction ($\wedge$)
`||`	disjunction ($\vee$)
`!`	negation ($\neg$)

Examples:

```
i>=0 && i<10
```

```
i==j || i+j==0
```

```
!(i==0)
```

Logical operators

- The `!` operator is unary while `&&` and `||` are binary
- They operate on integers
- Any value other than `0` is considered *true*; the value `0` is *false*
- The result of a logical operator is `0` or `1`

`!expr`	result 1 if expr has a value of 0
`expr1 && expr2`	result `1` if `expr1` and `expr2` are both non-zero
`expr1 || expr2`	result `1` if `expr1` is non-zero or `expr2` is non-zero (or both are non-zero)

In all other cases: the result is `0`

Order of evaluation

- The `&&` and `||` operators evaluate the left-hand side first and only then the right-hand side

- If the result can be determined by the value of the left-hand side, **the right-hand side will not be calculated**
  (this is sometimes called *short-circuit evaluation*)

  Example: `(i != 0) && (j/i > 0)`

  - The condition `i != 0` is evaluated first
  - If `i` is not `0`, then `j/i > 0` is evaluated
  - If `i` is `0` then the *conjunction is always false* and we don't evaluate `j/i > 0` (avoiding *division by zero*)

Precedence and associativity

- `&&` and `||` have lower precedence than the comparison operators and associate to the left
- `!` has equal precedence to unary `+` and `-` and associates to the right

Examples:

`i < j && k < m` is equivalent to `(i < j) && (k < m)`

`i < j && j < k && k < l` is equivalent to `((i < j) && (j < k)) && (k < l)`

`!i == 0` is **not** equivalent to `i != 0`

Care with if

- A common mistake is to replace `==` (equality) with `=` (assignment)

  `if (i == 0) ...` tests whether i is equal to 0

  `if (i = 0) ...` assigns 0 to i and then tests whether the result is different from 0 (which is always false)

  Recommendation: `gcc` warns of possible errors of this type by compiling with the `-Wall` option.

Care with else

- When we put an `if` inside an `else`, we have to be careful to match the `else` correctly:

```
if (y != 0)
   if (x != 0)
      result = x / y;
else
   printf("error: y equals 0\\n");
```

- The indentation suggests that the `else` associates with the outermost `if`
- But the rule in C is that the `else` associates with the nearest `if` (the inner one).

Care with else

- A correctly indented version would look like this:

```
if (y != 0)
   if (x != 0)
      result = x / y;
   else
      printf("error: y equals 0\n");
```

- To associate the else with the outer if we have to delimit the inner if using curly braces:

```
if (y != 0) {
   if (x != 0)
      result = x / y;
} else
   printf("error: y equals 0\n");
```

Recommendation: to avoid these problems always use curly braces in an if that contains another if