Introdução à Programação 2020/2021

António J. R. Neves
Daniel Corujo

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

an@ua.pt / dcorujo@ua.pt

http://elearning.ua.pt/

- Boolean expressions
  - Relational operators
  - Logical operators
  - Properties
- Conditional execution
  - If statement
  - If -- else statement
  - If -- else if -- else statement
  - Switch statement

- A **boolean expression** is an expression that is either **true** (in that case its value is 1) or **false** (its value is 0).

```
n == 5      # this is a boolean expression
6 == n      # this is another boolean expression
```

- When making decisions, the C language assumes that any non-zero value means **true**, and that a zero value means **false**.
- **Relational operators** produce boolean results (an integer that is either 0 or 1):

```
x == y      # x is equal to y
x != y      # x is not equal to y
x > y       # x is greater than y
x < y       # x is less than y
x >= y      # x is greater than or equal to y
x <= y      # x is less than or equal to y
```

1. There are three **logical operators**:
   - **`expr1 && expr2`** (and logical operator)
   - **`expr1 || expr2`** (or logical operator)
   - **`! expr1`** (not operator)
2. Example:

```
x >= 0 && x < 10 // x is between 0 (inclusive)
                 //   and 10 (exclusive)

0 <= x && x < 10 // same thing
```

3. When in doubt about the precedence of the mathematical and logical operators, use parentheses around an expression:

```
( (x >= 0) && (x < 10) )
```

- The && and || logical operators evaluate expr2 **only** when the result cannot be inferred from the value of expr1
- For the && operator, if expr1 is false then the result of the && logical operator is false, irrespective of the value of exp2; in that case expr2 **is not evaluated**.
- For the || operator, if expr1 is true then the result of the || logical operator is true, irrespective of the value of exp2; in that case expr2 **is not evaluated**.
- Remember these properties:

```
x == y    <=>    !  (x != y)    <=>    y == x
x != y    <=>    !  (x == y)    <=>    y != x
x > y     <=>    !  (x <= y)    <=>    y < x
x <= y    <=>    !  (x > y)     <=>    y >= x
```

- Almost all programs need to make decisions that depend on the values of some of the program's variables. **Conditional statements** give us this ability.
- The simplest form is the **`if`** statement:

```
if(x > 0)
  cout << "x is positive";
```

- The expression inside the mandatory parentheses after the **`if`** is called the condition [the C++ standard calls it the controlling expression].
- The statement (or block) gets executed if the condition is true. If not, the statement or block is skipped.
- There is no limit on the number of statements that can appear in the block, but there has to be at least one.

- A second form of the **`if`** statement offers two possibilities (one executed when the expression is true and one when it is false).

```
if(x % 2 == 0)
  cout << "x is even";  // executed when x % 2 == 0 is true
else
  cout << "x is odd";  // executed when x % 2 == 0 is false
```

- Sometimes there are more than two possibilities and we need more than two branches:

```
if (x < y)
  cout << "x is smaller than y";
else if(x > y)
  cout << "x is larger than y";
else
  cout << "x and y are equal";
```

- When the action we wish to perform contains several statements, it is necessary to put the statements inside a statement block (statements surrounded by brackets).

```
if(x % 2 == 0){
  // executed when x % 2 == 0 is true
  cout << "x is even";
  x = x / 2;
}
else{
  // executed when x % 2 == 0 is false
  cout <<"x is odd";
  x = 3 * x + 1;
}
```

- One conditional can also be nested within another.

```
if(x == y)
  cout << "x and y are equal";
else {
  if(x < y)
    cout << "x is less than y";
  else
    cout << "x is greater than y";
}
```

The indentation is optional but it makes the code much more legible.

- Although the use of the brackets makes the structure apparent, nested conditionals become difficult to read very quickly.
- Logical operators often provide a way to simplify nested conditional statements.

- Transformations may simplify the layout of the code or make it easier to understand:

```
if (Cond1)
  if (Cond2)
    Block1
  else
    Block2
else
  Block3
```

```
if (! Cond1)
  Block3
else
  if (Cond2)
    Block1
  else
    Block2
```

```
if (! Cond1)
  Block3
else if (Cond2)
  Block1
else
  Block2
```

- A **switch** statement allows an integer expression to be tested for equality against a list of values.

```
switch(integer_expression)
{
   case constant_integer_expression :
      statement(s);
      break; // optional
   case another_constant_integer_expression  :
      statement(s);
      break; // optional
   /* you can have any number of case statements */
   /* it is an error to repeat a value           */
   default : // if it is not one of the other values ...
             // (can be placed anywhere, although putting it
             //    in the and makes more sense)
     statement(s);
     break; // optional
}
```

- The expression used in a switch statement must have an integer value (a char value is automatically converted to an integer value)
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- The number after the case keyword must be a **constant** known at compile time.
- If the break statement is missing, the next case will also be executed.

switch - example

deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

```
#include <iostream>
using namespace std;
int main(void)
{
   /* local variable definition */
   char grade = 'B';

   switch(grade) {
      case 'A' :
         cout << "Excellent!\n";
         break;
      case 'B' :
      case 'C' :
         cout << "Well done\n";
         break;
      case 'D' :
         cout << "You passed\n";
         break;
      case 'F' :
         cout << " Try again\n";
         break;
      default :
         cout << "Invalid grade\n";
         break;
   }

   cout << "Grade is " << grade;

   return 0;
}
```

Consider the following code:

```
if (x >= 0)
  y = x;
else
  y = -x;
```

Instead of using an **`if`** statement, it is possible to write this code in the following much more compact form:

```
y = (x >= 0) ? x : -x;
```

In general, the value of the expression

```
cond ? expr1 : expr2
```

is **`expr1`** if **`cond`** is true, and is **`expr2`** if **`cond`** is false; **`cond`**, **`expr1`** and **`expr2`** are general expressions (pay attention to the precedence of arithmetic operators, use parenthesis if in doubt)