Control Statements

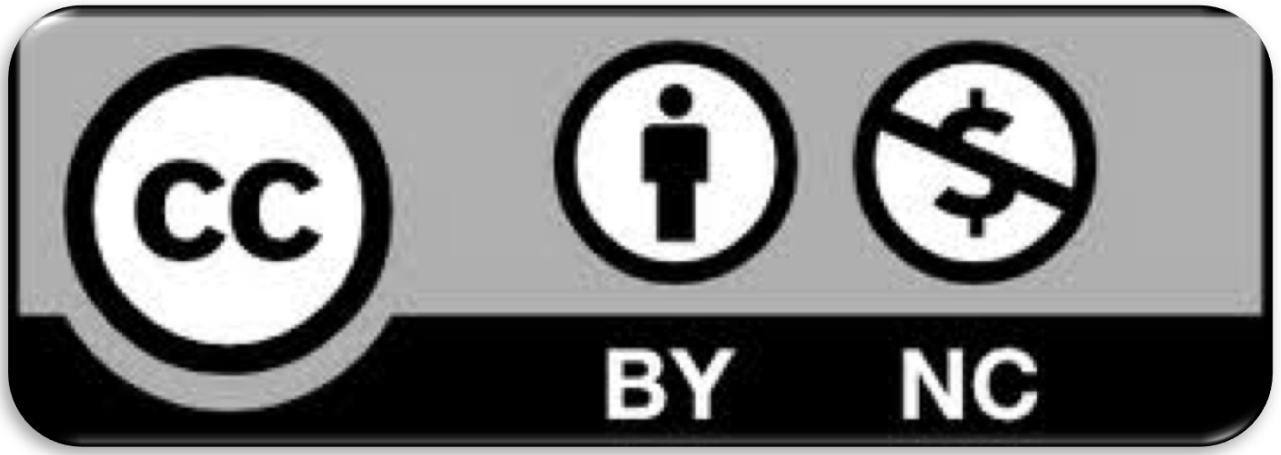
Alessandro Savino
Politecnico di Torino (Italy)

alessandro.savino@polito.it

www.testgroup.polito.it

License Information

This work is licensed under the Creative Commons BY-NC License



To view a copy of the license, visit:
http://creativecommons.org/licenses/by-nc/3.0/legalcode

Disclaimer

- **We disclaim any warranties or representations as to the accuracy or completeness of this material.**
- **Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.**
- **Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.**

Goal

- This lecture presents a global overviews of problems and issues related to flow control statements and their proper usage

Prerequisites

- **Basic knowledge of C programming language**

Homework

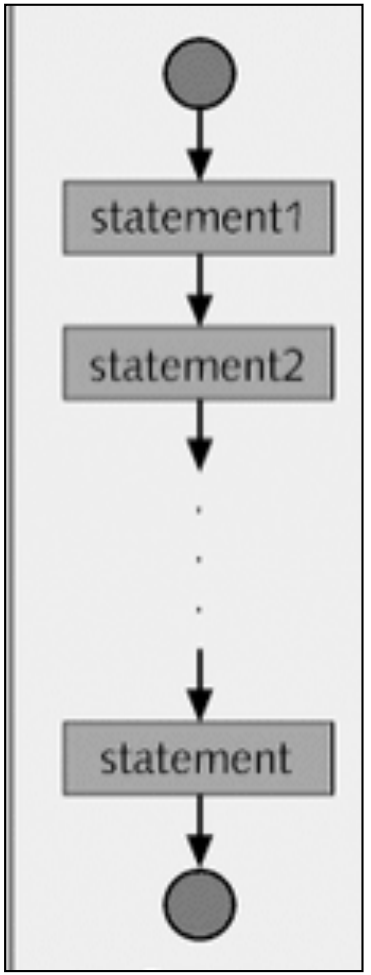
- **Applies to ALL labs**

Outline

- **Control Structures**
- **Relational Operators**
- **Logical (Boolean) Operators**
- **Logical Expressions**
- **Bitwise Operators**
- **Statements: IF and IF-ELSE**
- **Structures: SWITCH**
- **Operators: COMMA**
- **Functions and constructs: EXIT, ASSERT**

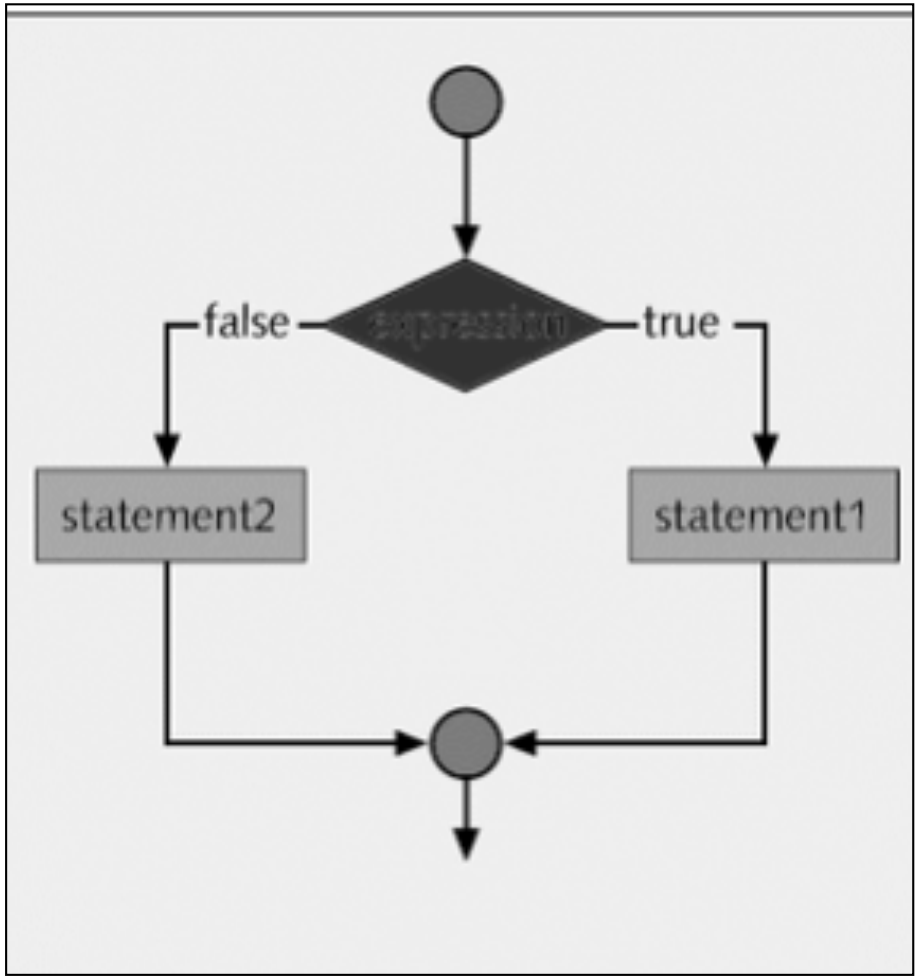
Control structures

- **Statements can be executed in sequence**
- **One right after the other**
- **No deviation from the specified sequence**



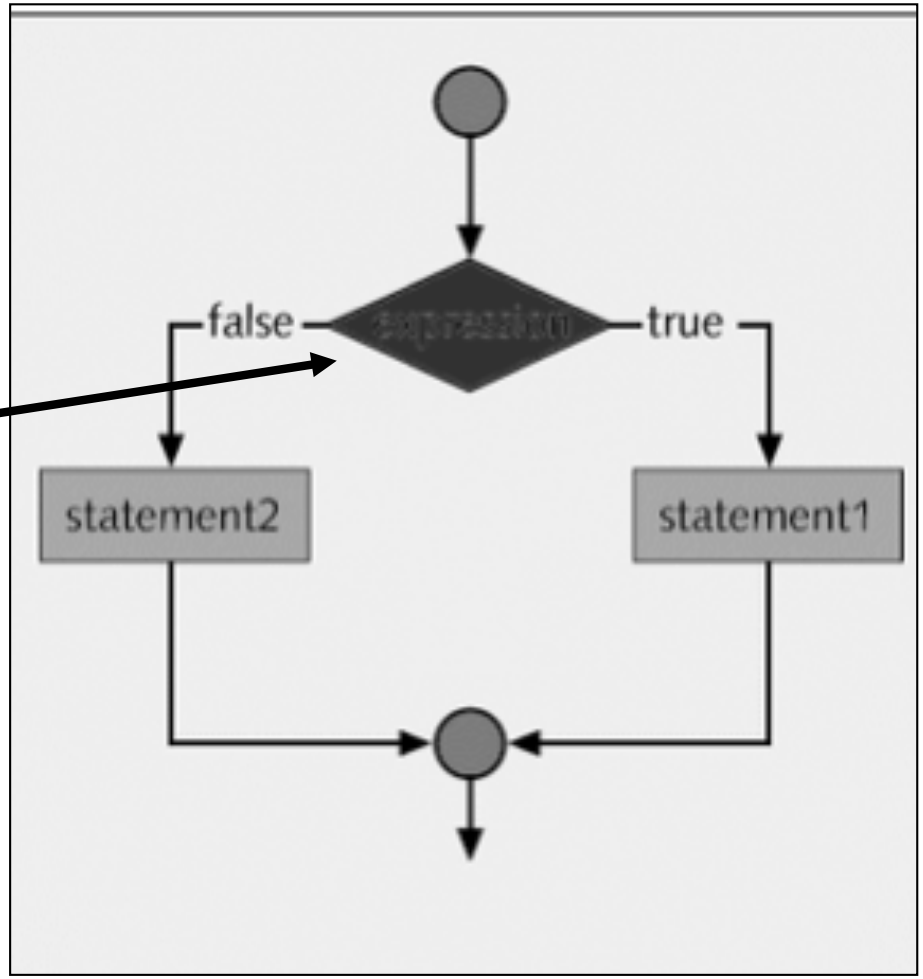
Control structures

- **A selection structure can be used**



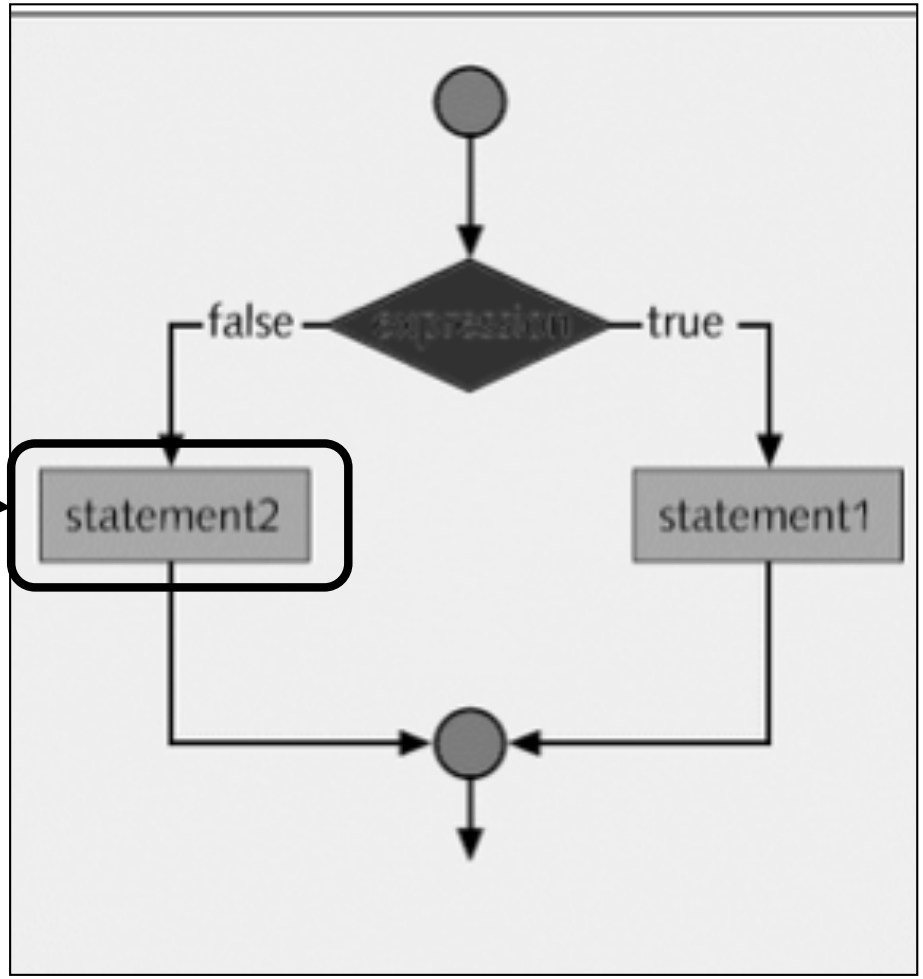
Control structures

- **A selection structure can be used**
- **Which statement is executed depends on whether the expression is true or false**



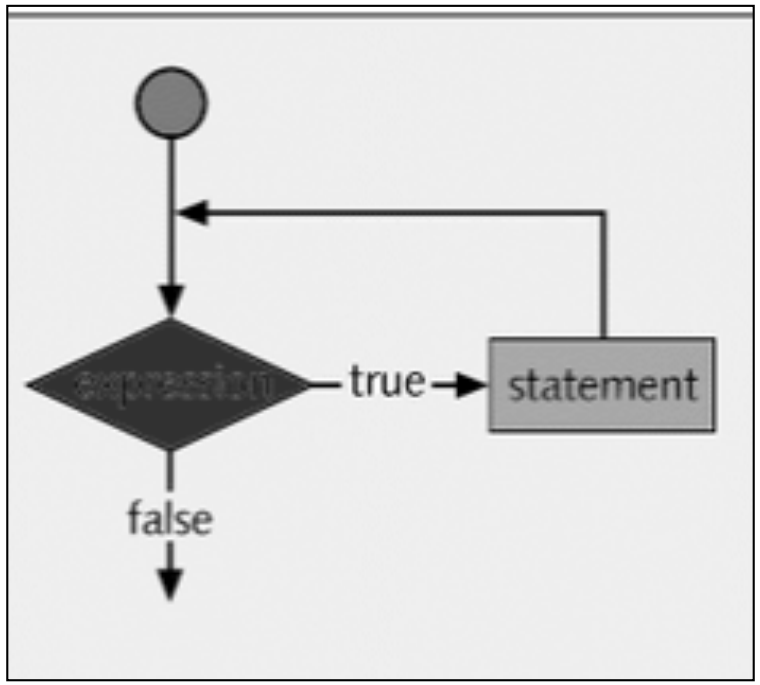
Control structures

- **A selection structure can be used**
- **Which statement is executed depends on whether the expression is true or false**



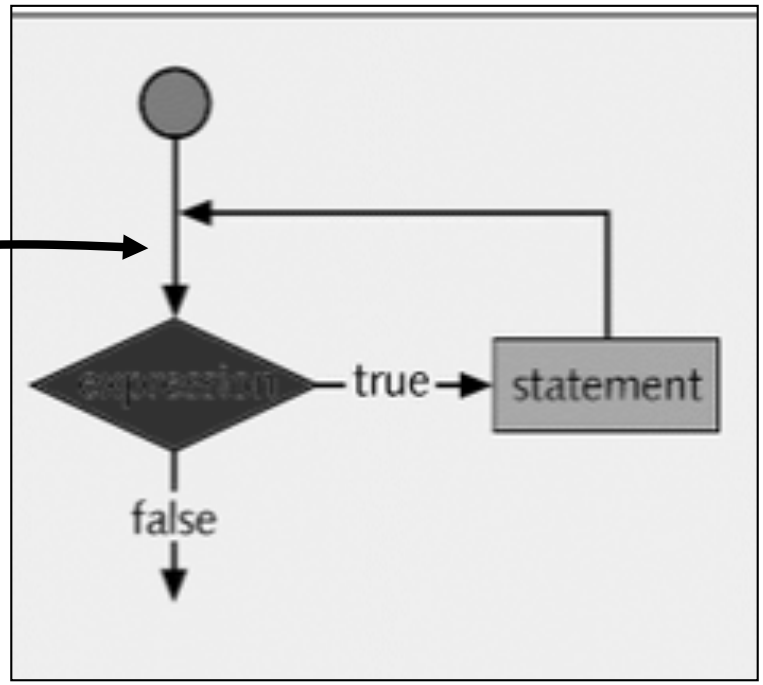
Control structures

- **Statements can be repeated**



Control structures

- **Statements can be repeated**
- **The number of repetitions depends on when the expression turns false**



Relational operators

- **The expressions which determine**
  - **Selection and**
  - **Repetition**

  } **are usually comparisons**
- **Comparisons are done with relational operators**

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

Relational operators

- **The expressions which determine**
  - **Selection and**
  - **Repetition**

  } **are usually comparisons**
- **Comparisons are done with relational operators**

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

Beware of mistaking the assignment = for the equality ==

Relational operators

Examples:

Expression	Meaning	Value
`8 < 15`	**`8` is less than `15`**	**`true`**
`6 != 6`	**`6` is not equal to `6`**	**`false`**
`2.5 > 5.8`	**`2.5` is greater than `5.8`**	**`false`**
`5.9 <= 7.5`	**`5.9` is less than or equal to `7.5`**	**`true`**
`6 == 5.99999`…	6 is **equal** to 5.9999…	**`???`**

Relational operators

Given

```
string str1 = "Hello";  string str2 = "Hi";
string str3 = "Air";
string str4 = "Bill";
string str5 = "Big";
```

Expression	Value
`str1 < str2`	true
`str1 > "Hen"`	false
`str3 < "An"`	true
`str1 == "hello "`	false
`str3 <= str4`	true

Determine the values of these comparisons using variables

Logical (Boolean) Operators

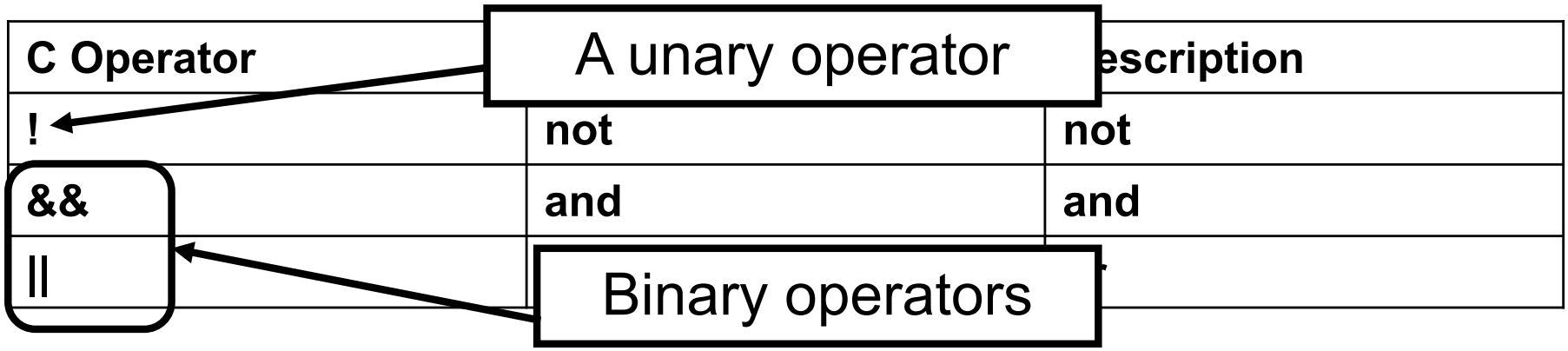
- **Logical or Boolean operators enable you to combine logical expressions**

C Operator	C++ Alternative	Description
!	not	not
&&	and	and
||	or	or

- **Operands must be logical values**
- **The results are logical values (true or false)**

Logical (Boolean) Operators

- **Logical or Boolean operators enable you to combine logical expressions**



C Operator		escription
!	not	not
&&	and	and
||		

- **Operands must be logical values**
- **The results are logical values (true or false)**

Logical (Boolean) Operators

- **The `&&` operator (logical and)**
  - **If both operands are true, the result is true**
  - **If either or both operands is false, the comparison is false**
- **The `||` operator (logical or)**
  - **If either or both of the operands are true, the comparison is true**
  - **The comparison is false only if both operands are false**
- **The `!` operator (logical not)**
  - **The not operator reverses the logical value of the one operand**

Logical Expressions

- **We must know the order in which to apply the operators**

```
12 > 7 || 9 * 5 >= 6 && 5 < 9
```

Order of Precedence

Highest

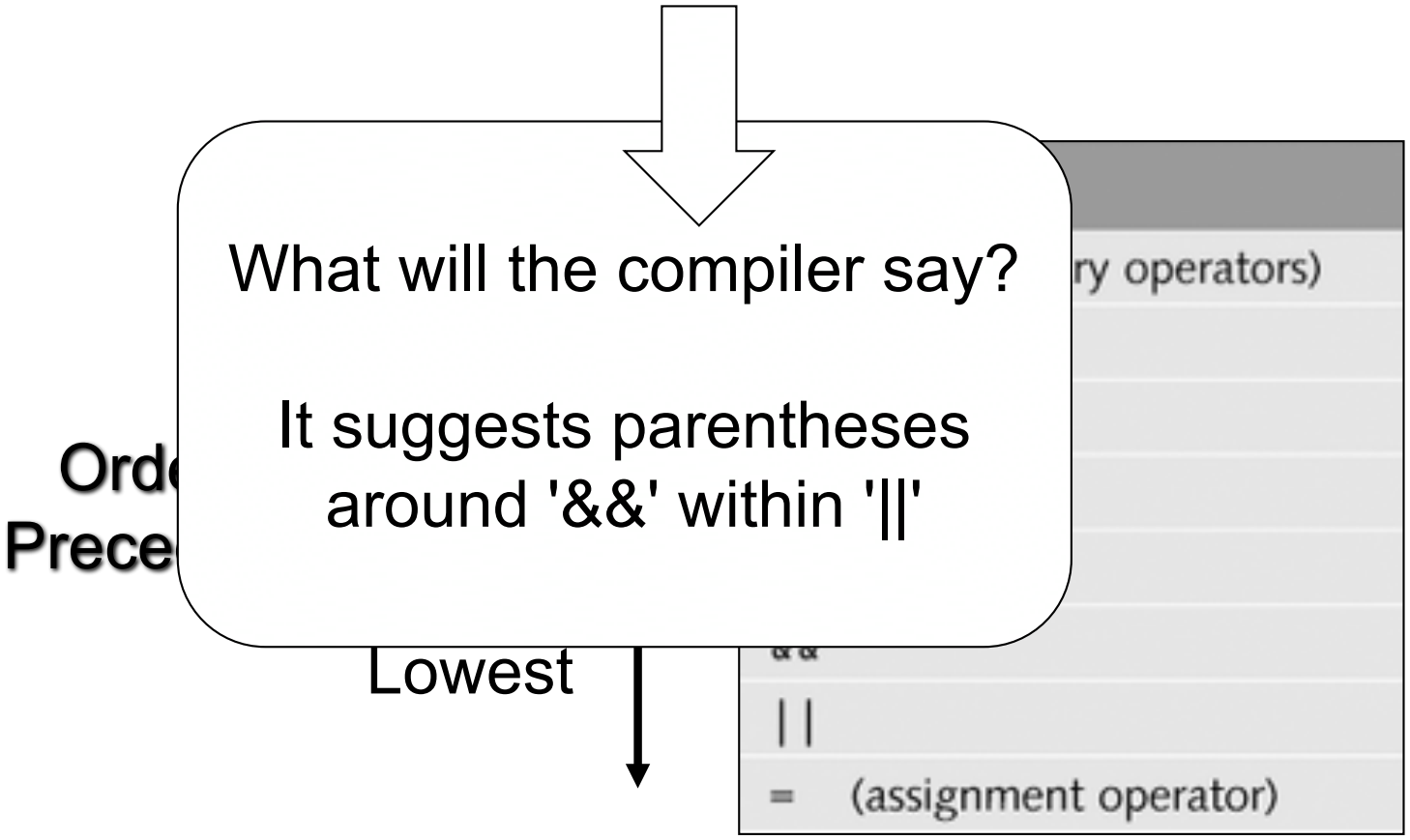
Lowest

Operators
!, +, – (unary operators)
*, /, %
+, –
<, <=, >=, >
==, !=
&&
||
= (assignment operator)

Logical Expressions

- **We must know the order in which to apply the operators**

```
12 > 7 || 9 * 5 >= 6 && 5 < 9
```



Logical Expressions

- **Pay attention!!!**

- **What happens if we use () ???**

  !0&&0||0

  !(0&&0)||0

Order of Precedence

Highest

Lowest

Operators
!, +, – (unary operators)
*, /, %
+, –
<, <=, >=, >
==, !=
&&
||
= (assignment operator)

Bitwise Operators

- **C and C++ have also bitwise operators**
  - **they help you manipulate each bit of a variable.**
    - **most of the time fixed point/integer variables**

```
01001000 AND
10111000 =
--------
00001000
```

Bitwise Operators

- **They require left and right operand**

```
var1/value1 <bitwise op.> var2/value2
```

C Operator	C++ Alternative	Description
&	bitand	and
|	bitor	or
~	compl	1's complement
!	not	not
^	xor	xor
<<		left shift
>>		right shift

Bitwise Operators

- **They require left and right operand**

C Operator	C++ Alternative	Description
&	bitand	and
>>		right shift

they can be either a variable or a explicit value
(e.g., var1, 0x3, 5, etc.)

Bitwise Operators

- **The result is not permanent if you forget to assign it somehow**

```
var = var1/value1 <bitwise op.> var2/value2
```

C Operator	C++ Alternative	Description
&	bitand	and
|	bitor	or
~	compl	1's complement
!	not	not
^	xor	xor
<<		left shift
>>		right shift

Bitwise Operators

- **The result is not permanent if you forget to assign it somehow**

```
var = var1/value1 <bitwise op.> var2/value2
```

this must be a variable!

C Operator	C++ Alternative	Description
&	bitand	and
<<		left shift
>>		right shift

Bitwise Operators

- **In particular, the shift operations expect the right operand to represents the number of bits to be shifted**

```
0xf << 0x3 // produces 0x78
```

C Operator	C++ Alternative	Description
&	bitand	and
|	bitor	or
~	compl	1's complement
!	not	not
^	xor	xor
<<		left shift
>>		right shift

Bitwise Operators

- **In particular, the shift operations expect the right operand to represents the number of bits to be shifted**

```
0xf << 0x3 // produces 0x78
```

C Operator	C++ Alternative	Description
&	bitand	and
<<		left shift
>>		right shift

0xF → 0000 1111
left shift by 3 bits → 0111 1000
7 8

IF statement

- **C++ has “two versions” of if statements**

IF statement

- **Syntax**

```
if ( logicalExpression )
        statement;
```

- **Example**

```
if (x < 5 )
   cout << "low value for x";
```

IF statement

- **Syntax**

```
if ( logicalExpression )
       statement;
```

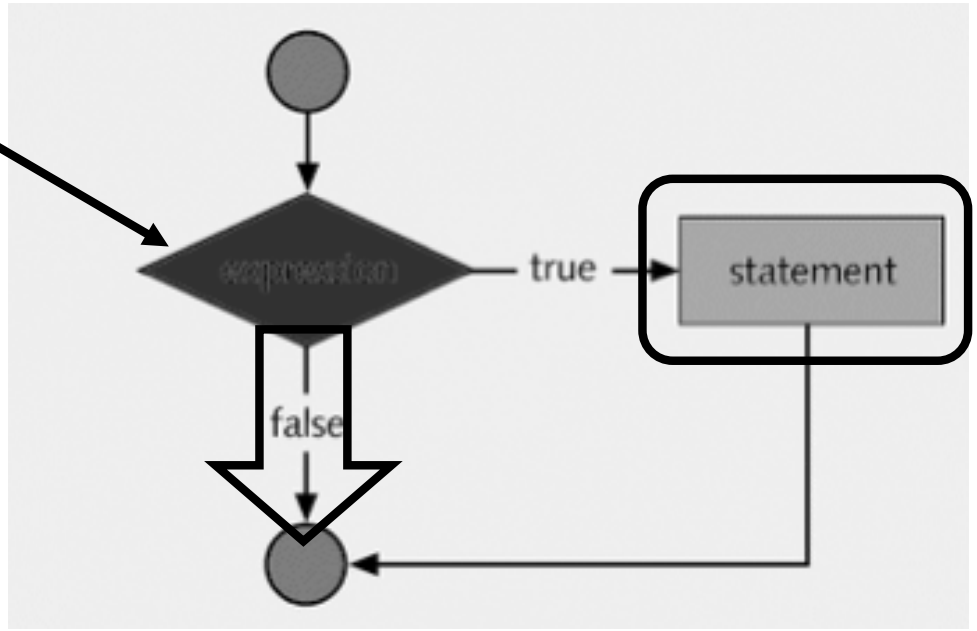
- **Example**

```
if (x < 5 )
{
   cout << "low value for x";
}
```

More readable code

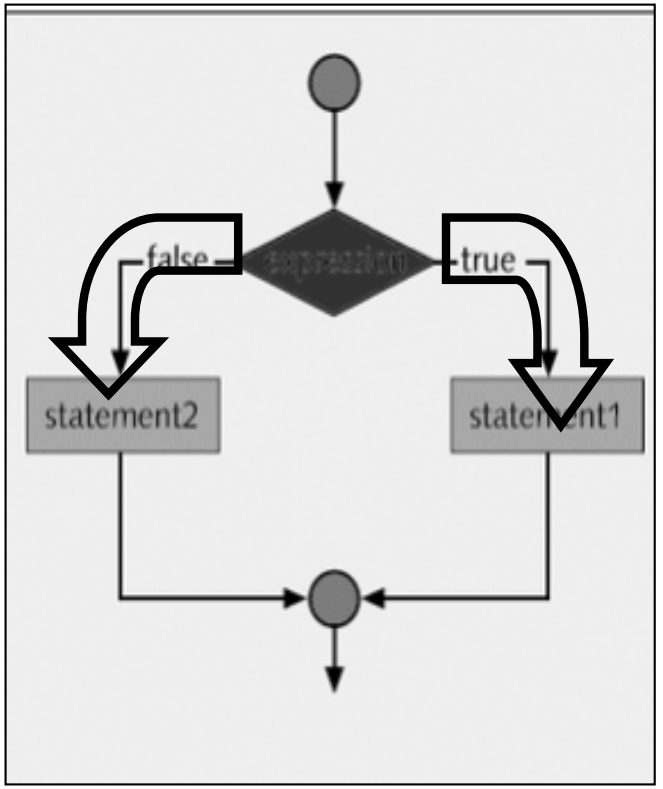
IF statement

- C++ has “two versions” of if statements
- In this version, the condition is checked
  - If the expression is true, the statement is executed
  - If it is false, nothing happens



IF – ELSE statement

- **Also possible a two way selection**
- **If the expression is true, statement1 is executed**
- **Otherwise statement2 is executed**



IF – ELSE statement

- **Syntax**

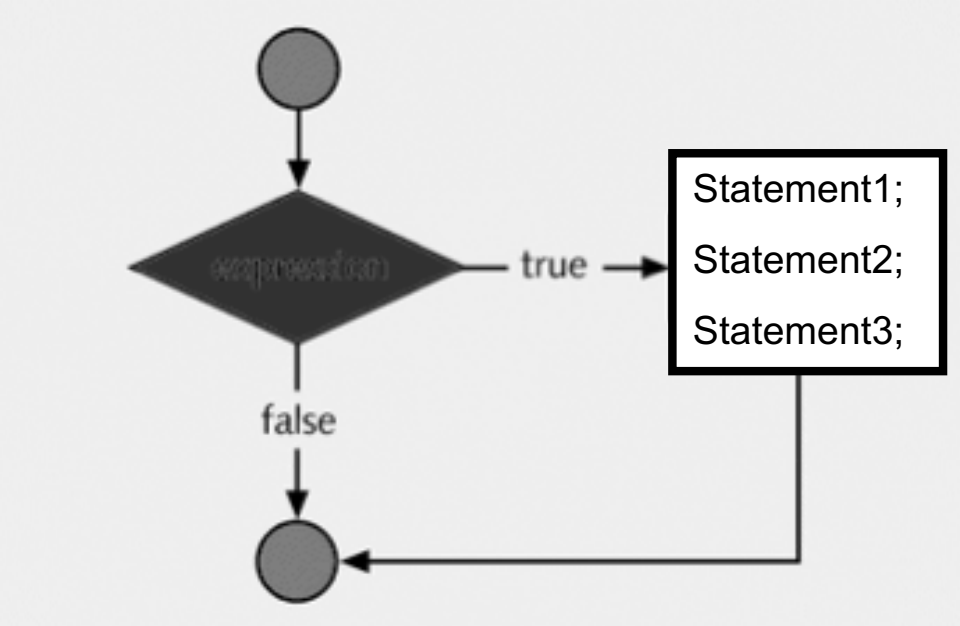
```
if (condition)
    statement1;
else
   statement2;
```

- **Example**

```
if (x < 5) cout << "low x";
     else    cout << "high x";
```

Compound Statement

- **Consider the need for multiple statements to be controlled by the `if`**
- **This is called a compound statement**
- **Group the statements in curly brackets**



Compound Statement

- **Example**

```
if (x < 5)
   {
      x = x + 10;
      cout << x;
   }
```

The compound statement

- **Note the use of indenting and white space in the source code for readability. But for readability ONLY !!! For the compiler this code is equal to:**

```
if (x < 5){x = x + 10;cout << x;}
```

Nested IF

- **Syntax calls for a "statement" after the if (…)**
- **That statement can be any kind of statement**
- **It can be another if statement**

```
if (x < 7)
  if (y > 5)
    cout << "hi mom";
```

Dangling ELSE

- **How to determine which `if` the `else` goes with?**

- **Example:**

```
if (abs (x - 7))
  if (x < 7) cout << "x approaches 7 from left";
   ?   else
          cout << "x approaches 7 from the right";
  ? else
          cout << "x not close to 7";
```


Rel. 23/03/2017

Dangling ELSE

- **How to determine which `if` the `else` goes with?**

- **Example:**

```
if (abs (x - 7))
  if (x < 7) cout << "x approaches 7 from left";
   ?   else
          cout << "x approaches 7 from the right";
  ? else
          cout << "x not close to 7";
```

Rule : An ***else*** goes with the closest ***unmatched if***

Dangling ELSE

- **Rule : an `else` goes with the closest unmatched `if`**

```
if (x < y)

   if (y > 3) cout << "message about y > 3";
        else  cout << "message about x and y";
```

- **Consider … <u>how</u> do you <u>force</u> an `else` to go with a previous `if`?**

Use { curly brackets }
to nest the statements

```
if (x < y)
 { if (y > 3) cout << "message about y > 3"; }
        else  cout << "message about x and y";
```

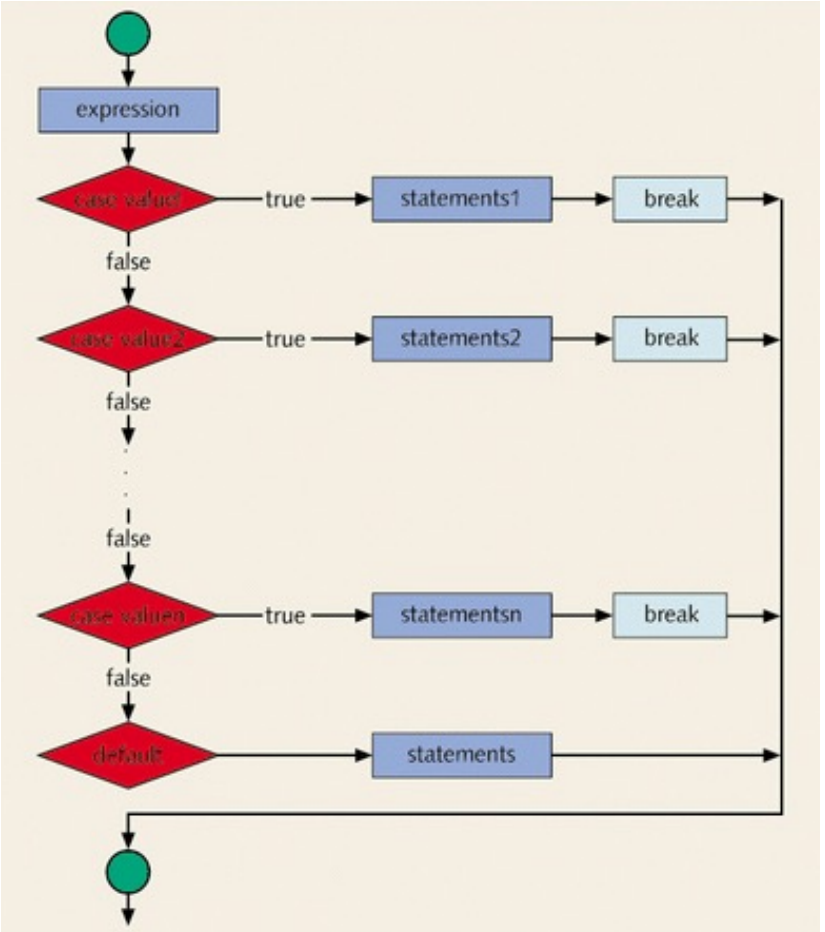

Rel. 23/03/2017

Multiple selections

- **Contrast**
  - **A sequence of**
    `if … else if …` **statements**
  - **A sequence of separate `if` statements**
- **What happens in each case when it is the first if condition that is true?**
  - **`if … else if` sequence will jump out of the structure whenever match is found**
  - **sequence of separate `if`'s – each `if` is checked, no mater where the match is.**

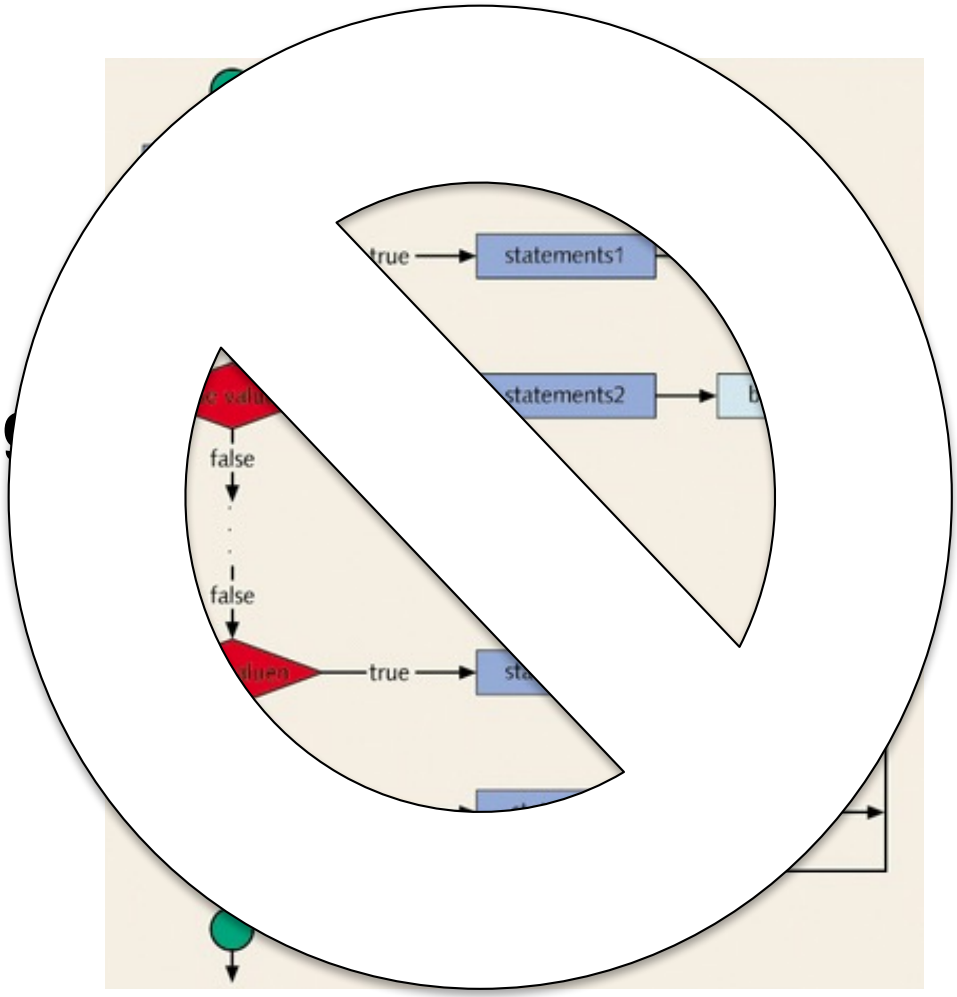
Multiple selections

- **Recall the current branching capability provided by the `if ( … )` statement**
- **Only branches two ways**
- **We desire a better way to do multiway branching**



Multiple selections

- **Recall the current branching capability provided by the `if ( … )` statement**
- **Only branches two ways**
- **We desire a better way to do multiway branchin**



SWITCH statement

- **C++ provides the `switch` statement**

```
switch (choice) {
case 1 : do_option_one(); break;
case 2 :
case 3 : do_2_3_a ();
         do_2_3_b (); break;
default : do_something_else ();  }
```

SWITCH statement

- **Value of the switch expression matched with one of the labels attached to a branch**

```
switch (choice) {
case 1 : do_option_one(); break;
case 2 :
case 3 : do_2_3_a ();
         do_2_3_b (); break;
default : do_something_else ();  }
```

- **The statement(s) with the match get executed**

SWITCH statement

- **Switch expression => the expression in parentheses whose value determines which switch label is selected**
  - **cannot be floating point**
  - **usually is `int` or `char`**
  - **can only control equality**
  - **each case must be different**

- **Identifiers following `case` must be constants**

SWITCH statement

```
switch (choice) {
case 1 : do_option_one();
         break;
case 2 :
case 3 : do_2_3_a ();
         do_2_3_b ();
         break;
default : do_something_else ();

}
// next statement
```

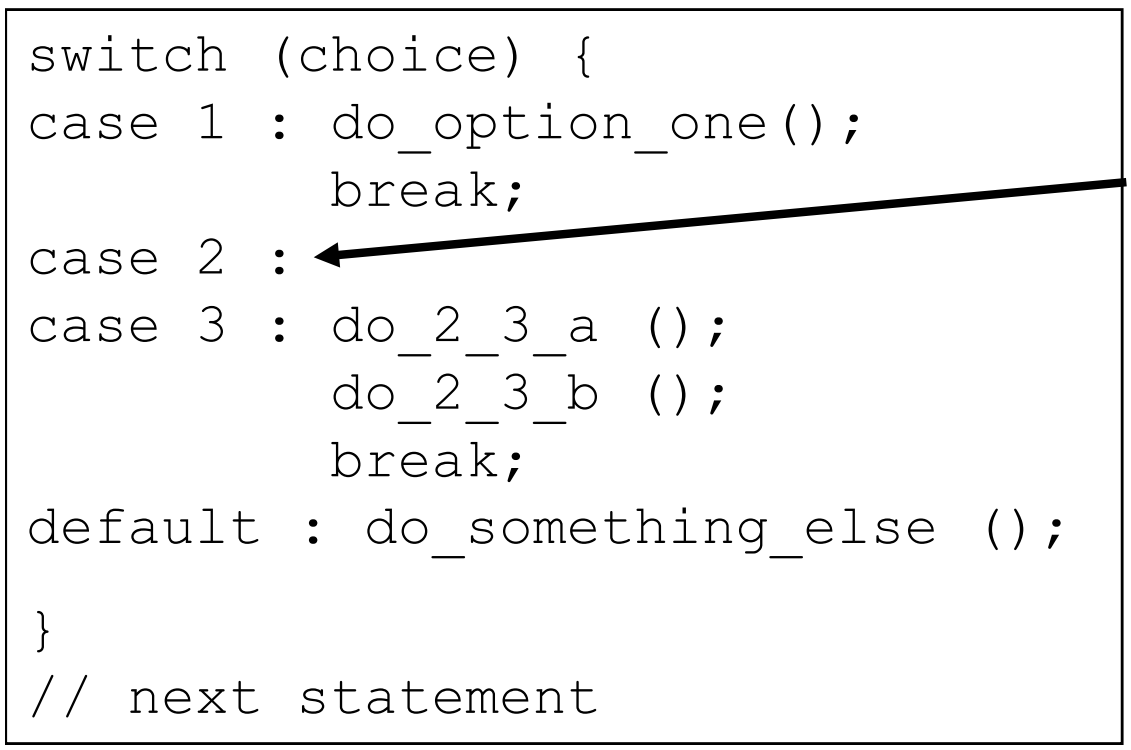
- **The `break` causes control to be shifted to first statement after the switch statement**

The `default` statement is executed if the value of the switch expression is NOT found among switch labels

SWITCH statement

```
switch (choice) {
case 1 : do_option_one();
         break;
case 2 :
case 3 : do_2_3_a ();
         do_2_3_b ();
         break;
default : do_something_else ();

}
// next statement
```



- **Pay attention!!! C++ inherits a powerful, functionality from C: since there is not a break statement, when choice is 2 BOTH case 2 and case 3 will be executed!**

WHILE loop

- **The `while` loop iterates until the expression is true**

```
while (expression) {
      Processing statement
}
```

WHILE loop

- **What happens if expression is never true?**

```
int b = 5, c = 0;
while (b < 3){
    ++c;
}
```

WHILE loop (example)

```
#include <iostream>

void main(void)
{
int x = 0 ;
while(x <= 10)
      {
      cout << ++x << endl;
      }
}
```

- **This will print the numbers 0 to 10 on successive lines.**

Rel. 23/03/2017

DO - WHILE loop

- **The `do - while` loop is useful in conditions where a certain set of processing statements needs to be performed at least once.**

```
do {
     Processing statements
} while (expression);
```

DO - WHILE loop (example)

```
#include <iostream>


void main(void){
int x = 0 ;
do{
     cout << ++x << endl;
} while(x > 0);
}
```

Nested loops

- **To enable multiple layers of iteration.**

```
while(expression){
 while(expression){
   statements
 }
}
```

```
do {
 do {
   statements
 }while(expression);
}while(expression);
```

FOR loop

- **The general syntax of the for statement is :**

```
for (initialization; test; update) {
     Processing statements
}
```

- **The initialization is an assignment statement that sets the loop control variable(s), before entering the loop.**
- **The test is a relational expression, which determines, when the loop will exit.**
- **The update defines how the loop control variable(s) change(s), each time the loop is executed.**

FOR loop

- **The three sections of the for loop must be separated by a semicolon (;) .**

- **The statement, which forms the body of the loop, can either be a single statement or a compound statement.**

- **The `for` loop continues to execute as long as the conditional test evaluates to true. When the condition becomes false, the program resumes on the statement following the `for` loop.**

FOR loop (example)

```
#include <iostream>

void main(void)
{
   int x;
   for (x=0; x<10; x++)
   {
      cout << x << endl;
   }

}
```

- **This will print the numbers 0 to 9 on successive lines.**

Rel. 23/03/2017

FOR loop (example)

```
#include <iostream>

void main(void)
{
   int x;
   for (x=0; x<10; x++)
   {
      cout << x << endl;
   }

}
```

x does exist here!

- **This will print the numbers 0 to 9 on successive lines.**

FOR loop (example)

```
#include <iostream>

void main(void)
{
   for (int x=0; x<10; x++)
   {
      cout << x << endl;
   }

}
```

- **This will print the numbers 0 to 9 on successive lines.**

x does NOT exist here!

EXIT construct

- **The function `exit()` is used to terminate a program immediately. An exit() is used to check if a mandatory condition for a program execution is satisfied or not. This function NEVER returns.**

- **Provided within cstdlib**

- **The general form of an exit() is:**

```
exit(int return_code)
```

EXIT Construct (example)

```
#include <iostream>
#include <stdlib>

void main(void){
   int val = 1 ;
   char c_entry ;
   while(val <= 50){
      cout << "Val = " ;
      ++val ;
      cout << "Enter E to exit system
        immediately";
      cin >> c_entry ;
      if(c_entry == 'E' || c_entry == 'e')
        exit(0) ;
   }
cout << "Exiting system..." ;
}
```

EXIT Construct (example)

```
#include <iostream>
#include <stdlib>

void main(void){
   int val = 1 ;
   char c_entry ;
   while(val <= 50){
      cout << "Val = " ;
      ++val ;
      cout << "Enter E to exit system
        immediately";
      cin >> c_entry ;
      if(c_entry == 'E' || c_entry == 'e')
        exit(0) ;
   }
cout << "Exiting system..." ;
}
```

Not reached if exit!

EXIT Construct (example)

```
#include <iostream>
#include <stdlib>

void main(void){
   int val = 1 ;
   char c_entry ;
   while(val <= 50){
      cout << "Val = " ;
      ++val ;
      cout << "Enter E to exit system
        immediately";
      cin >> c_entry ;
      if(c_entry == 'E' || c_entry
        exit(0)
   }
cout << "Exiting system..." ;
}
```

Return code:
0 or EXIT_SUCCESS indicate success;
EXIT_FAILURE otherwise

Rel. 23/03/2017


Малые Автюхи, Калинковичский район, Республики Беларусь