Lecture 3:

Decision Structures: *if* statement

Prof. Shervin Shirmohammadi
University of Ottawa

Decision Structures

- Recall that computers execute instructions one at a time
  - In fact most pseudo-code and C instructions are broken down into smaller machine instructions
  - A program can be viewed as a list of ordered instructions to be executed by the CPU
  - Would be nice to have the CPU make decisions on executing certain instructions
- Enter **decision structures**:
  - Allows the computers to skip certain instructions, based on specific decisions. More precisely: jump to instructions based on a value of TRUE or FALSE (in reality a value of 1 or 0)
  - Such decisions are taken by evaluating logical expressions.

Logical Expressions

- Expressions involving comparison operators such as “equal”, “greater than”, or logical operators such as AND, OR, and NOT.
- Like arithmetic expressions, the computer evaluates logical expressions to obtain a value: FALSE (0) or TRUE (1).
- Truth tables for the logical operators:

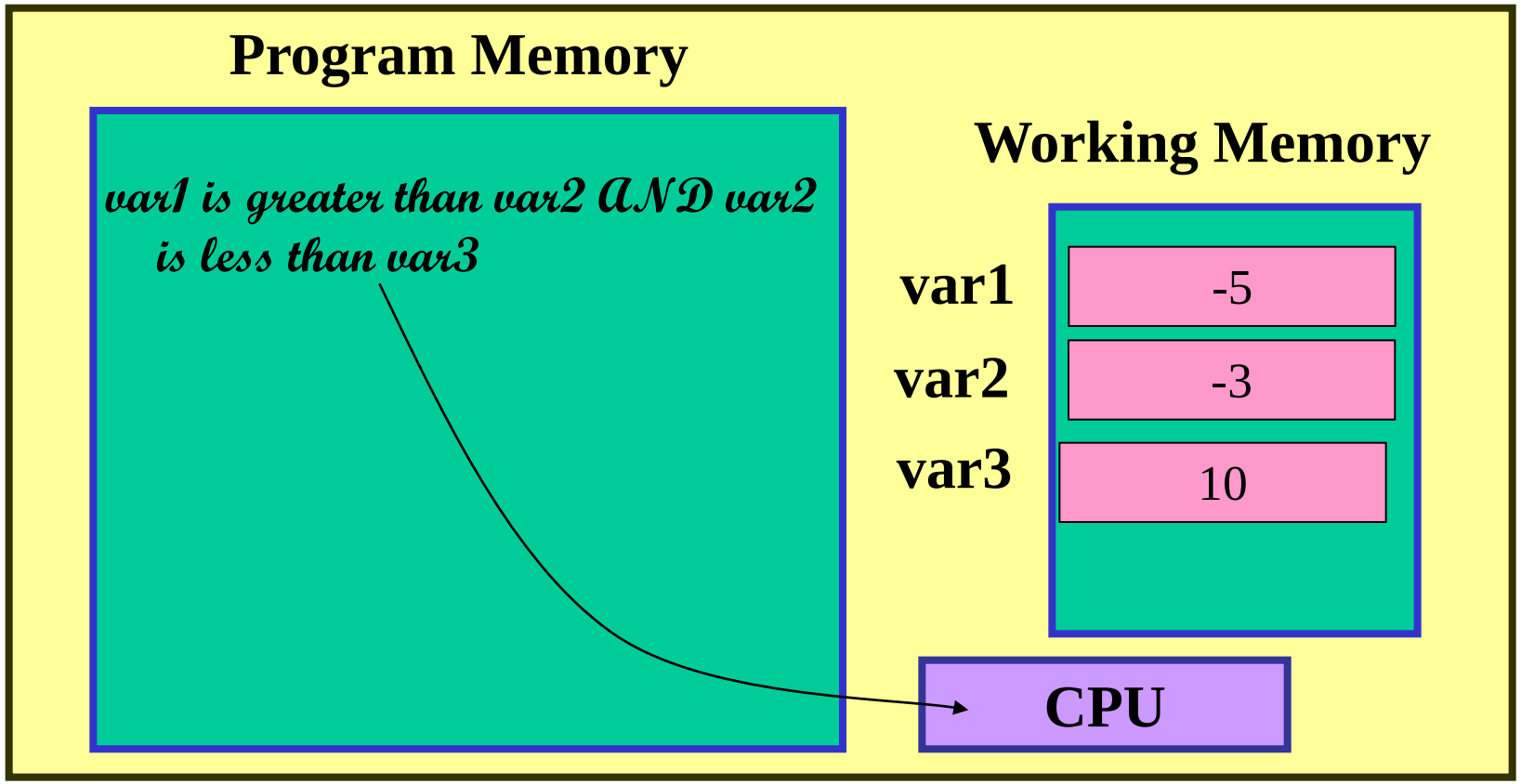
X	Y	X AND Y	X OR Y	NOT X
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

- In pseudo-code, shall write out the comparison and logical operators as in the following example:

var1 is greater than var2 AND var2 is less than var3

Logical Expressions (continued)

- How does the CPU evaluate the pseudo-code given below?
- What is done with the resulting value?
  - As we shall see later, such results can be used in a decision structure



Relational, Equality and Logical Operators in C

- Relational operators:

Operation	C Operator	example C Logical Sub-expression
x>y	>	`x > y`
x<y	<	`x < y`
x≥y	>=	`x >= y`
x≤y	<=	`x <= y`

- Equality operators:

Operation	C Operator	example C Logical Sub-expression
x=y	==	`x == y`
x≠y	!=	`x != y`

- Logical operators:

Operation	C Operator	example C Logical Sub-expression
AND	&&	`x && y`
OR	||	`x || y`
NOT	!	`!x`

  - The `!` operator is a unary operator while the `&&` and `||` operators are binary operators.

Rules of Logical Operator Precedence in C

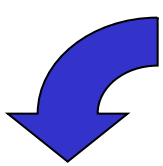
- In a logical expression,
  - the logical not `!` operator has highest precedence,
  - the relational operators have the next highest precedence,
  - then come the equality operators,
  - then the logical `&&` operator,
  - and finally the logical `||` operator which has the lowest precedence.
- The following table summarizes in order of precedence the C operators that we have seen to date:

`()`	left to right	
`- ! ++ -- (type)`	right to left	**unary operators**
`* / %`	left to right	
`+ -`	left to right	
`< <= > >=`	left to right	
`== !=`	left to right	
`&&`	left to right	
`||`	left to right	
`= += -= *= \= %=`	right to left	

Logical Expressions in C

- The result of a logical expression can only be true or false.
- In C, a logical expression that evaluates to **false** has the integer value **0** and a logical expression that evaluates to **true** has the integer value **1**.
- Here are a few examples of valid logical expressions in C:

```
int k = -3, j = -4;
float a = 5.5, b = 1.5;
a < 10.5 + k;
a + b >= 6.5;
k != a - b;
b - k > a;
a < 10 && a > 5;
k / j == 0.75;
```



Result

1
1
1
0
1
0

- In C, logical expressions always evaluate to integers 1 or 0 (for true or false).

- In C, a non-zero integer value (`1`, `2`, `-1`...) is interpreted as being true in a logical expression. Suppose that variables `x` and `y` are of type `int`; below are the truth tables for our logical operators in C, operating on the variables `x` and `y`.
  - Truth table for the logical and:

X	Y	X and Y
T	T	T
T	F	F
F	T	F
F	F	F

In a C expression

x	y	x && y
10	5	1
3	0	0
0	2	0
0	0	0

  - Truth table for the logical or:

X	Y	X or Y
T	T	T
T	F	T
F	T	T
F	F	F

In a C expression

x	y	x || y
150	6	1
99	0	1
0	67	1
0	0	0

  - Truth table for the logical not:

X	not X
T	F
F	T

In a C expression

x	!x
3	0
0	1

Example 1

- Given integers x, y, and z, write the following expressions:
  - z is a positive number
  - y is less than x or larger than z
  - z is not equal to x
  - y is equal to 5
  - z is even

The Decision Structure

- Upon evaluation of a logical expression, we can specify a set of instructions to execute when the expression is TRUE and another set when the expression is FALSE:

  If logical_expression

  Set of instructions to execute if logical_expression is true

  Otherwise

  Set of instructions to execute if logical_expression is false

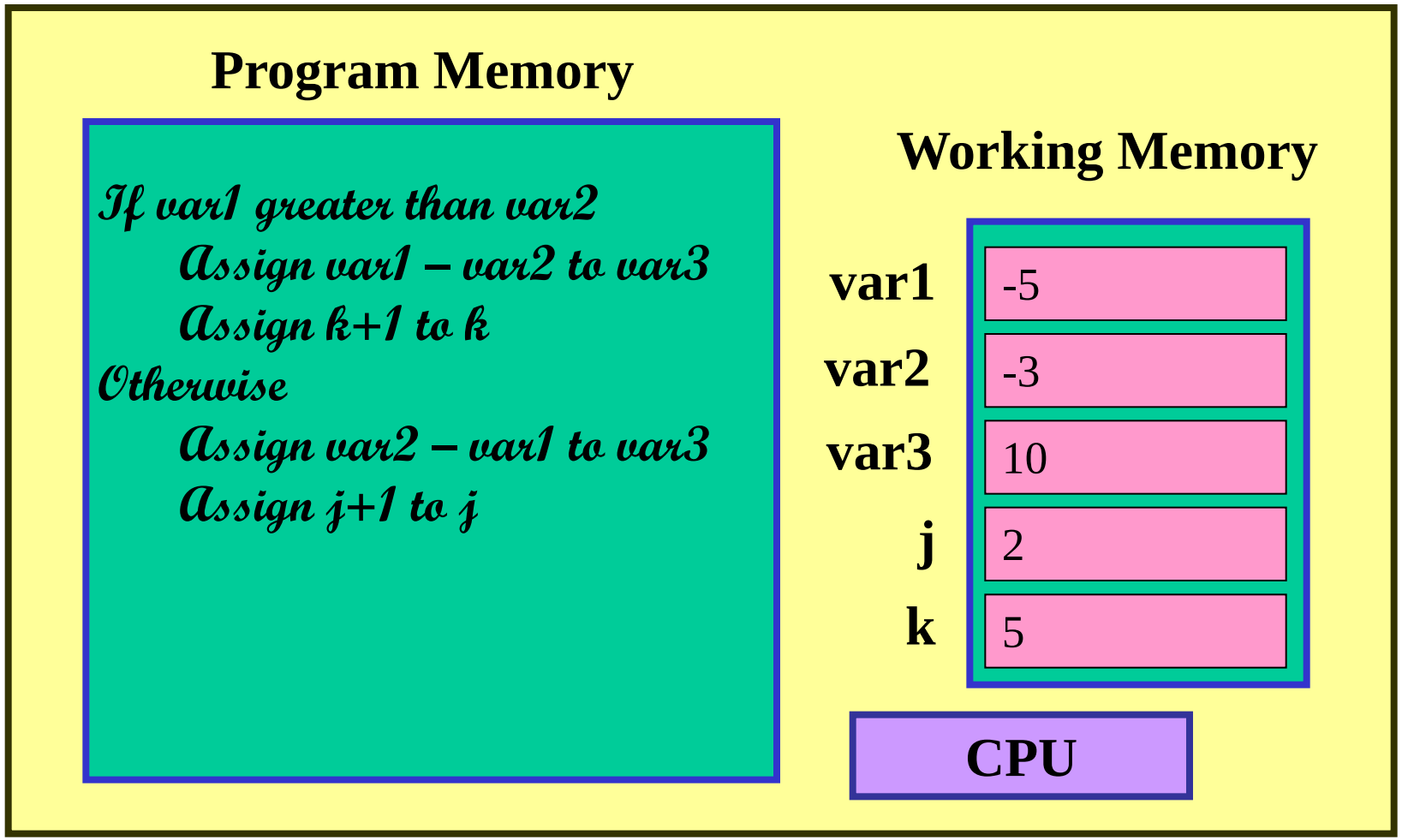
  – In pseudo-code, all instructions at the same indentation level make up a set of instructions (see example on next slide)

  – Not necessarily so in a C program!

- Nesting decision structures
  – The decision structure is considered a single instruction, so it is possible to include another decision structure as part of the set of instructions within a decision structure, as w shall see later.

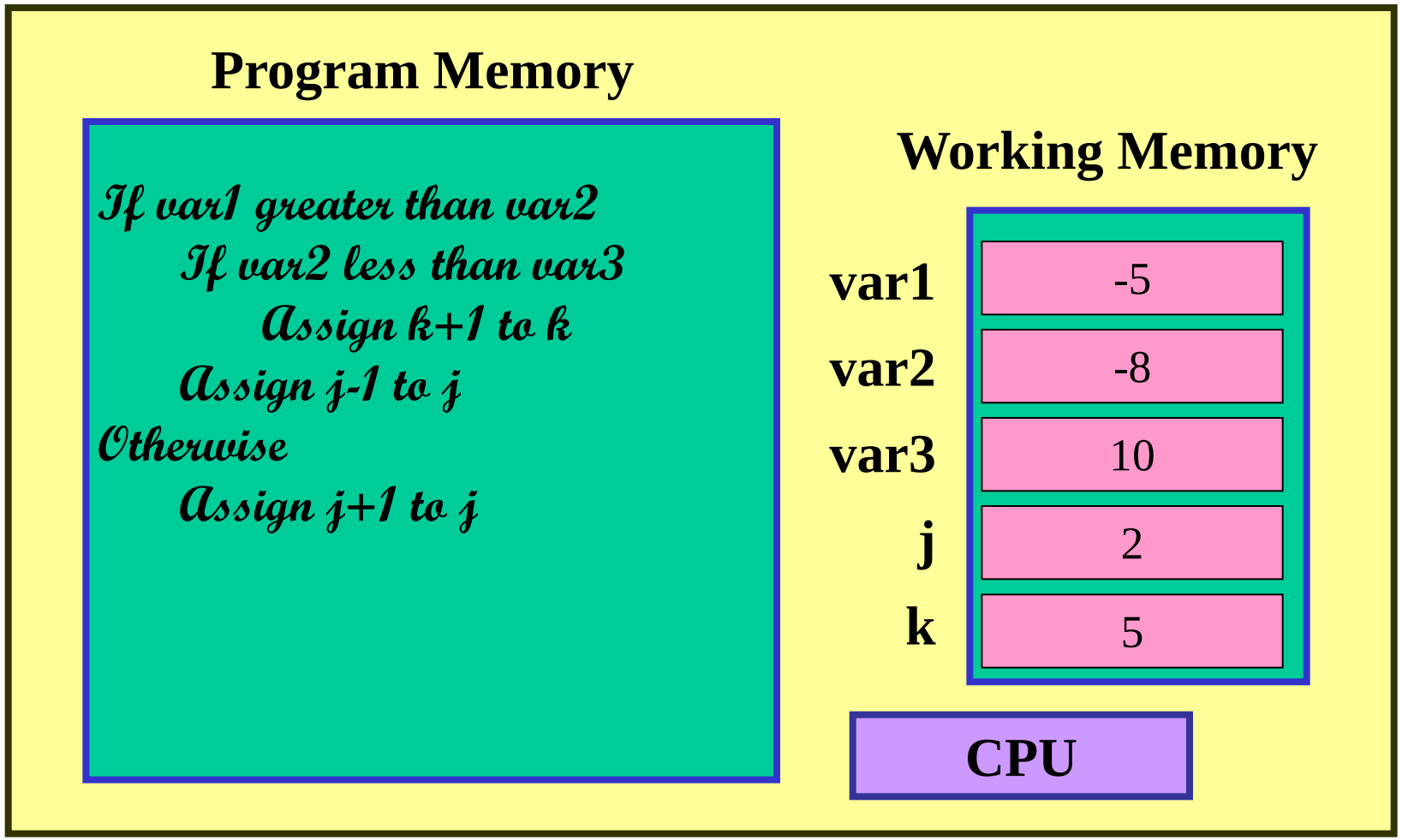
Decision Structure: pseudo-code example

- How does the CPU execute the pseudo-code given below?



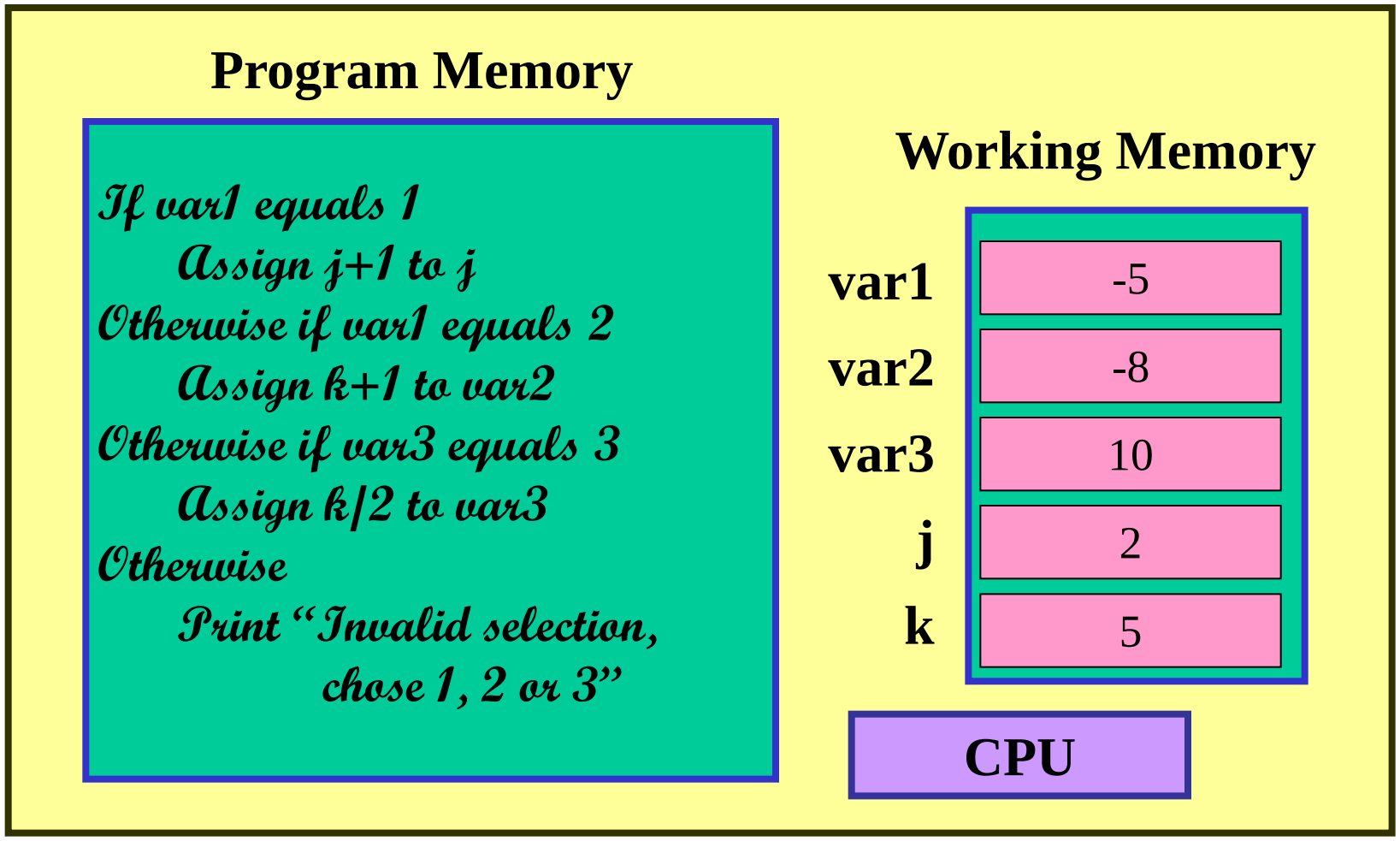
Nested Decisions: pseudo-code example

- How does the CPU execute the pseudo-code given below?
  - Note that the "otherwise" portion of a decision structure is optional



Nested Decisions: the otherwise-if structure

- The following form provides a short-hand notation for multiple nested decision structures.



Decision structures in C

Four decision structures exist in C:

- The `if` statement
  - The basic decision structure with a set of instructions to be executed if a logical expression is TRUE, otherwise do nothing.
- The `if-Else` statement
  - If a logical expression is TRUE do something, otherwise do something else
- The Nested `if-Else` statement
  - Multiple logical expressions
- The `switch` statement
  - An alternate form of the `Else-if` structure when testing values.

The *if* Decision Structure

- The `if` decision structure allows us to execute instructions only if a logical expression is true.
- Syntax for single instruction:

```
if (logical expression)
    instruction;
```

Careful: there is no `;` here.

The C instruction directly below the `if ()` is executed only if `logical expression` evaluates to true.

- Syntax for multiple instructions:

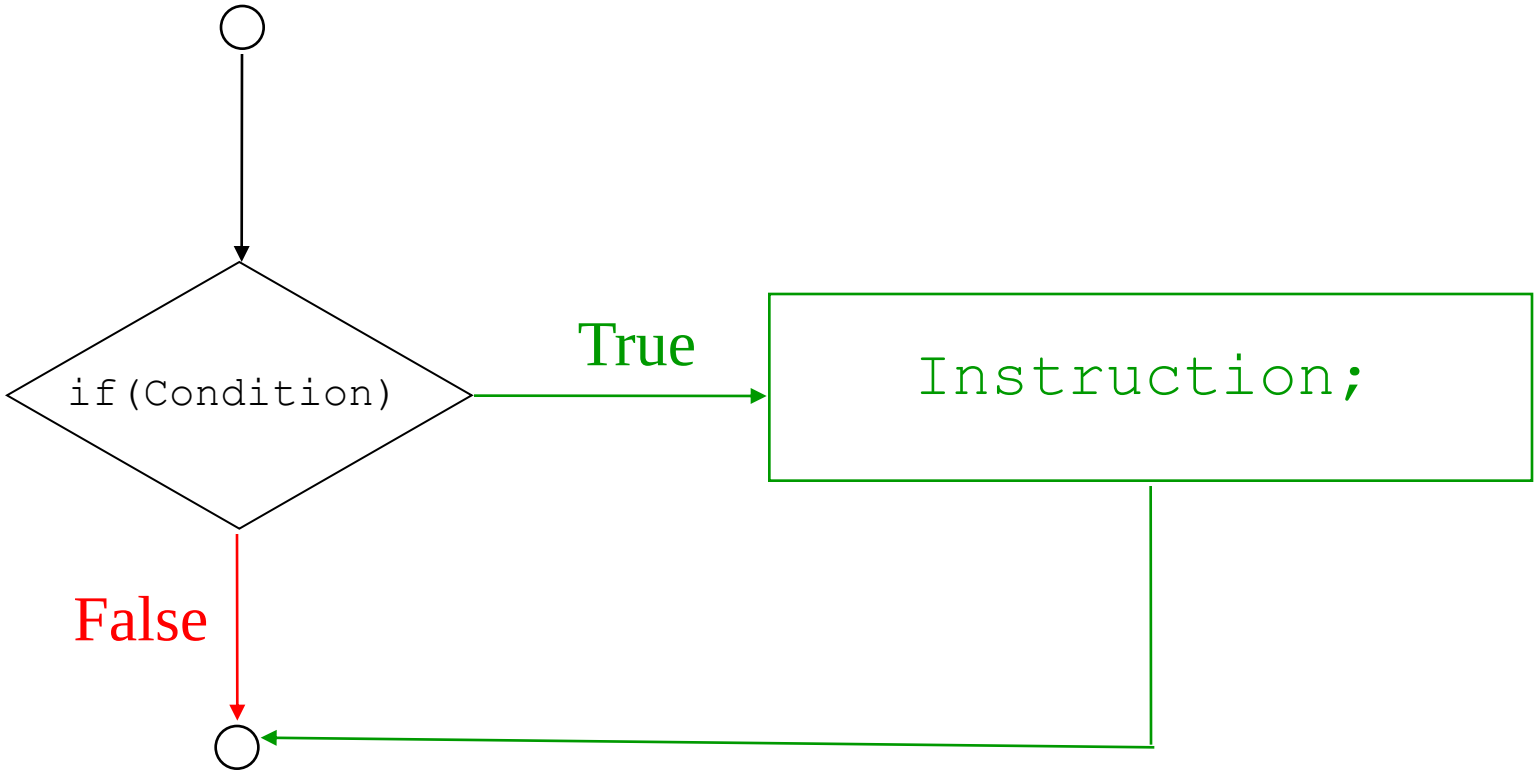
```
if (logical expression)
{
    instruction 1;
    instruction 2;
    :
    instruction n;
}
```

The C instructions delimited by a pair of `{}` directly below the `if ()` are executed only if `logical expression` evaluates to true.

- It is very good practice to indent the instructions under the `if` in order to help visualize the logical structure of the program. Using `{}` to isolate a single instruction is possible and can also help clarify the code.

- A **flowchart** is a graphical description of a language structure, algorithm, or small program.
  - A flowchart is a convenient way to visualize logical flow.
  - Diamonds are used to represent decisions and simple instructions are shown as boxes.

- Flowchart of the `if` decision structure:



- Using the `if` structure:

```
if (a < 10.0)
    a = a + 1.0;
```

  - or:

```
if (a < 10.0)
{
    a = a + 1.0;
    b = b + 1.0;
}
```

If `a` was declared as:
`float a = 5.5;`
then in all of these examples the logical expression evaluates to true and the commands isolated by the `if`s are executed.

- We can also have nested `if`s:

```
if (a > 1.0)
   if (a < 10.0)
      a = a + 1.0;
```

  - The above can also be written using logical operators:

```
if (a < 10.0 && a > 1.0)
   a = a + 1.0;
```

Example 2

- Write a program that enters a float number representing the grade of a student and print out whether the grade is a valid value or not.

  Hint: the grade is invalid if outside the range [0,100]

Example 3

- Write a program which input three numbers and output the message "sorted" if the numbers are in ascending order and output "not sorted" otherwise.