

Chapter 3: Conditional Instructions

1. Introduction

A program consists of instructions. Most of these instructions are executed in the order in which they appear. Once an instruction is executed, it moves to the next instruction (sequential). Instructions are often separated by a semicolon ";". However, some instructions modify the program flow, known as control structures. For example: conditional structures, loops, function calls, and unconditional jump instructions. The conditional structure comes in three types: the simple conditional structure (if then), the complex conditional structure (if then else), and the multiple-choice structure (switch).

2. The Simple Conditional Structure "if then"

In programming, we often encounter cases where we need to decide whether an instruction should be executed or not based on whether a condition is true or false. For example, in a dessert recipe, you might find instructions like: If you have almonds, add them to the recipe, or if you like lemons, add a little more. The program's execution flow will change as the input changes. To express conditions in programming, we use the if...then test, which is the simplest conditional instruction. It consists of two parts:

- Condition: A Boolean expression with a value of either true or false.
- Block of instructions: executed if the condition is true, or ignored if the condition is false.

2.1. Syntax:

Algorithm	C
if Condition then Block of instructions EndIf The rest of the instructions	if (Condition) { Block of instructions } The rest of the instructions

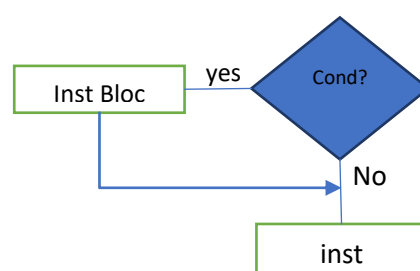
The words "if," "then," and "EndIf" (or "fi") are reserved words in the algorithm. The same applies to "if" in C. In the algorithm, the condition is always between "if" and "then," while in C, it is always enclosed in parentheses (). To build the condition, we use comparison operations (>, <, =, ≠, ...) and logical operations (and, or, not, ...).

Instructions belonging to the "if" in C are enclosed in curly braces {} which can be omitted if they contain only one instruction. (Optional {}). If we find a set of instructions after the "if," and we don't find the curly braces, then only the first instruction is associated with the condition, and the rest of the instructions will always be executed regardless of the condition. However, if there is more than one instruction inside the curly braces {}, then both curly braces are mandatory.

Note:

- In C, the Boolean type is represented by an int. False is represented by 0, and true is any number other than 0.
- There is no ";" after }.

2.2. Flowchart

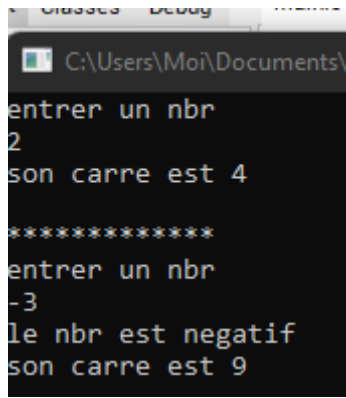


2.3. Execution

The execution process of the conditional instruction is performed by evaluating the condition, which results in a Boolean logical value. If the result is true, the block of instructions between "then" and "EndIf" in the algorithm, or between {} in C, is executed, followed by the rest of the program instructions. If the result is false, the instructions between "then" and "EndIf" are ignored, and the rest of the program instructions are executed directly.

2.4. Example

Write a program that reads an integer, then displays a warning if it's negative, and finally shows its square.

Algorithm	C	screen
<pre> algorithm root var x : integer begin write ("Enter a number: ") read (x) If x<0 then write ("nbr is negative ") End If write ("the square is " , x*x) end </pre>	<pre> #include <stdio.h> int main() { int x ; printf("Enter a number: \n") ; scanf("%d", &x) ; if (x<0) { // can be removed printf("nbr is negative \n") ; } printf("the square is %d" , x*x) ; } </pre>	

3. The Complex Conditional Structure "if then else"

In a simple conditional, "if" specifies what to do if the condition is true, but not what to do if it's false. However, sometimes it's necessary to decide what to do in both cases. This leads to the "if else" (if...then...else) structure, which is an extension of the simple "if." The complex conditional structure "if else" consists of three parts:

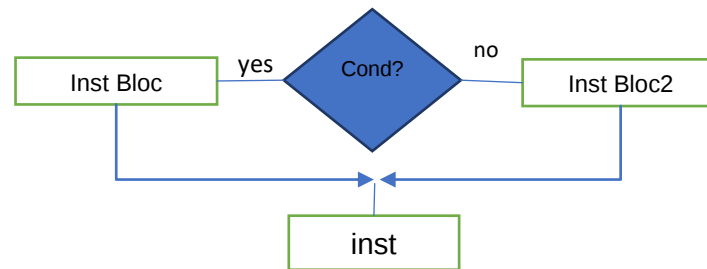
- Condition: A Boolean expression with a true or false value.
- First block of instructions: executed if the condition is true, or ignored if false.
- Second block of instructions: executed if the condition is false, or ignored if true.

3.1. Syntax:

Algorithm	C
<pre> If Condition then instruction block 1 else instruction block 2 End If The rest of the instructions </pre>	<pre> if (Condition) { instruction block 1 } else { instruction block 2 } The rest of the instructions </pre>

The word "Else" is a reserved word in the algorithm. The same applies to "else" in C. In C, {} can be omitted if it contains only one instruction. If there is a set of instructions after "if" or after "else," and curly braces are not found, it means that only the first instruction is related to "if" or "else."

3.2. Flowchart

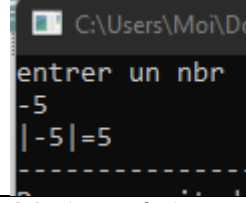


3.3. Execution

3.3. Execution The conditional instruction is executed by evaluating the condition, which results in a Boolean value. If the result is true, the first block of instructions between "then" and "else" in the algorithm, or between {} before "else" in C, is executed, followed by the rest of the program instructions. If the result is false, the second block of instructions between "else" and "EndIf" in the algorithm, or between {} after "else" in C, is executed, followed by the rest of the program instructions.

3.4. Example

Write a program that calculates the absolute value of an integer and displays it on the screen.

Algorithm	C	screen
<pre> algorithm absolute var x, y : integer begin write ("Enter a nbr: ") read (x) if x>=0 then y←x else y←-x End If write (" " , x , " =", y) end </pre>	<pre> #include <stdio.h> int main(){ int x, y ; printf("Enter a nbr:\n") ; scanf("%d", &x) ; if (x>=0) { // can be deleted y=x ; } else { // can be deleted y=-x ; } printf((" %d =%d", x, y)) ; } </pre>	 <pre> if (x>=0) y=x ; else y=-x ; </pre>

3.5. Conditional assignment in C

If we have a variable v takes one of the values v1 or v2 depending on condition b, i.e. :

```

if (b)
  v=v1 ;
else
  v=v2 ;

```

In this case, the “?:” can be used and its syntax is as follows:

condition ? expression_true : expression_false

- condition is a Boolean condition
- expression_true The expression returned if the condition is true.
- expression_false The expression returned if the condition is false

Example

```

v=b ? v1 :v2 ;
result = average >=10 ? "Admitted" : "Adjourned" ;

```

3.6. If-else extension

" If-else" can be used to test multiple conditions and to select the appropriate treatment for each case. For instance: to determine whether a student is accepted or not, there are several cases. Either they are accepted without compensation, or they are accepted with compensation, or they are accepted but with

debts, or they are postponed. To determine this, one must examine the averages of the first and second semesters (s1 and s2), the annual average (MA), and the total earned credits (Crd).

The solution

Algorithm	C
<pre> algorithm absolute var s1, s2, MA: real Crd : integer begin write ("Enter first and second semester averages ") read (s1,s2) write ("Enter annual average ") read (MA) write ("Enter total credits ") read (Crd) if s1>=10 and s2>=10 then write("admitted without compensation ") else if MA>=10 then write("admitted with compensation") else if Crd>=45 then write ("admitted with debts ") else write ("adjourned ") end if end if end if end </pre>	<pre> #include <stdio.h> int main(){ float s1, s2, MA ; int Crd ; printf("Enter first and second semester averages \n") ; scanf("%f%f", &s1, &s2) ; printf("Enter annual average \n") ; scanf("%f", &MA) ; printf("Enter total credits \n") ; scanf("%d", &Crd) ; if (s1>=10 && s2>=10) printf("admitted without compensation") ; else if (MA>=10) printf("admitted with compensation") ; else if (Crd>=45) printf("admitted with debts ") ; else printf("adjourned ") ; } </pre>

4. The Multiple-Choice Conditional Structure "switch"

To choose an action among multiple options, we use the "switch" statement. However, when dealing with more than two options, nested "if" statements can be used, resulting in nested "if" statements for each choice. This can make the program harder to read. The "switch" test is a special case of nested "if else" statements. It determines which block of code to execute based on the value of the tested variable. It is used when we have multiple outcomes and the condition is tested multiple times using the same variable. The "switch" statement is more readable and consists of:

- The expression to test, typically a variable.
- The values to test with corresponding blocks of instructions.
- An optional default block if there is no match with any value.

4.1. Syntax

Algorithm	C
<pre> case expression of val_1 : instruction block 1 val_2 : instruction block 2 ... val_n : instruction block n else another instruction block End case The rest </pre>	<pre> switch (expression) { case val_1 : instruction block 1 break ; ... case val_n : instruction block n; break ; default: another instruction block } The rest </pre>

The words "Case" "else" and "End Case" (or " EndCase") are reserved words in the algorithm. The same applies to "switch" "case" and "default" in C.

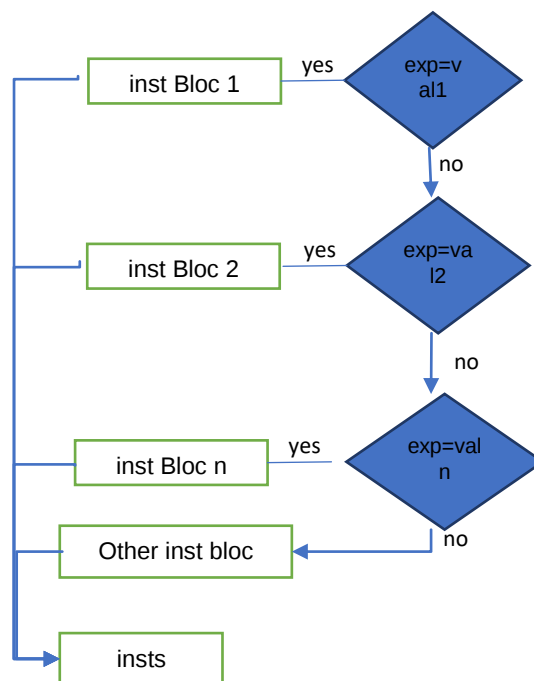
- **Expression:** An expression that calculates an integer or character value. It's typically a variable.
- **val₁, ..., val_n :** Values or constants of the same type as the expression.
- **Block of instructions:** One or more instructions executed if the expression value matches Value_i.

Note: switch is used instead of nested "if" we're going to test a single instruction or variable, of integer or character type, several times with constant values.

4.2. Rules regarding switch

- The curly braces {} of the switch and the parentheses () are necessary and cannot be omitted.
- Each Value_i must be different from the others. For example, writing "case 1" twice is illegal.
- Value_i can be placed in any order. However, it's recommended to order them in ascending order for better readability.
- A block of instructions can contain any number and type of instructions.
- The "break;" statement is optional. It's used to exit a switch immediately, moving the program flow out of the switch.
- The default block is optional. If no Value_i matches, the execution context will move to the default block. It should be the last case.

4.3. Flowchart



4.4. Execution

The "switch" statement is executed by evaluating the expression, then jumping to the block of instructions corresponding to the matched Value_i. After that block is executed, the execution will continue until it encounters a "break;" statement or reaches the end of the switch. If there is no match, the execution will move to the default block (if present), then continue with the rest of the program instructions.

The execution of "switch" in C slightly differs from the algorithm's "case". In C, after executing the block for Value_i, if no "break;" statement is encountered, the execution will continue with the subsequent block until a "break;" statement is reached. The execution will then move to the rest of the instructions outside the switch.

To make "switch" equivalent to "case algo" a "break;" statement should be added at the end of each block.

If multiple values share the same block of instructions, the algorithm can use a comma. In C, the first value should not have instructions or a "break;" statement. Assuming values 7 and 9 have the same treatment: Algorithm:

algorithm	C
7 , 9 : instruction block	<pre> case 7 : case 9 : instruction block break ; </pre>

4.5. Example

Write a program that reads an integer less than 10 and displays the corresponding English word on the screen.

Algorithm	C	The display
<pre> algorithm conversion var nb : integer begin write ("enter a nbr ") read (nb) case nb of 0 : write ("zero") 1 : write ("one") 2 : write ("two") ... 9 : write ("nine") else write ("not treated") End case end </pre>	<pre> #include <stdio.h> int main(){ int nb ; printf("enter a nbr \n") ; scanf("%d", &nb) ; switch (nb) { case 0 : printf("zero") ; break ; case 1 : printf("one") ; break ; ... case 9 : printf("nine") ; break ; default: printf("not treated") ; } return 0 ; } </pre>	

5. Branching Instructions

Branching is the process of moving between executed program instructions by the processor, where it performs a "jump" to a specific address instead of continuing to execute instructions sequentially. There are four instructions in C that can unconditionally modify the execution flow of a program: break, goto, continue, and return.

5.1. Break Statement

We've already seen it with "switch," where it ends the "switch" instruction, moving the flow to the first instruction after "switch." In the case of a nested "switch," it only exits the immediate enclosing "switch." It's also used to exit loops (covered in the next lesson). In this case, "break;" is usually within an "if."

Example

```

switch (grade){
    case 'A' :
    case 'a' : printf("excellent\n") ;
                break ;
    case 'b' : printf("good\n") ;
    case 'c' : printf("you can do better\n") ;
                break ;
    default : printf("try again\n") ;
}

```

- If grade contains the letter a or A, excellent.

- If it contains b 'it will appear good and you can do better
- If it contains the letter c, it only shows that you can do better.
- If it contains another character, try again.

5.2. Goto Statement

It transfers the program execution to a named instruction. This name, or "label," is preceded by a colon ":". Any instruction can be named with a valid identifier followed by a colon.

Label syntax: `label : instruction;`

where label is a valid identifier. Such as:

```
here : printf("zero") ;
```

Syntax for calling: To access this instruction from anywhere, use the following syntax:

```
goto label ;
```

where "label" is the name of the instruction, e.g.: To access the instruction "here" from anywhere, use:

```
goto here ;
```

Note:

- "case" and "default" are special naming methods used within a "switch."
- **Goto** can be used to repeat instructions without the need for loops.
- It's advisable not to use "goto" and labels extensively, as it makes the program difficult to understand and maintain for humans.

Example:

again :

...

goto again ;

5.3. continue Statement

It's used with loops to move the flow to the end of the loop and directly to the next iteration, without completing the loop instructions. It's typically within an "if."

Syntax `continue ;`

5.4. Return Statement

It's used to exit functions and return a result. (semester 2)

Syntax: `return expression ;`

Example: `return 0 ;`

As commonly used at the end of the main() function