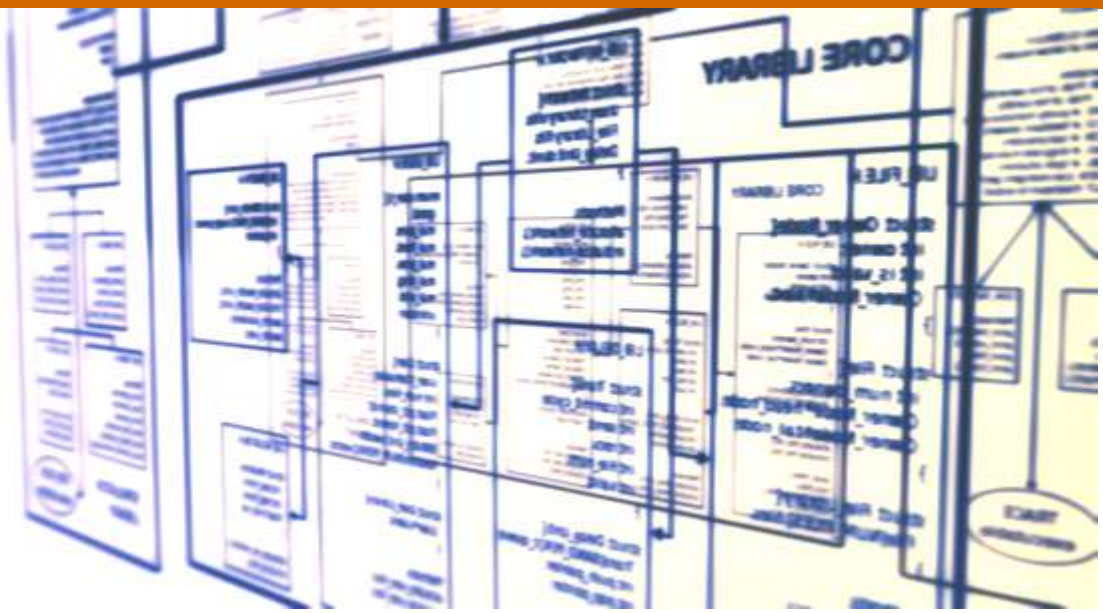
ALGORITHMS



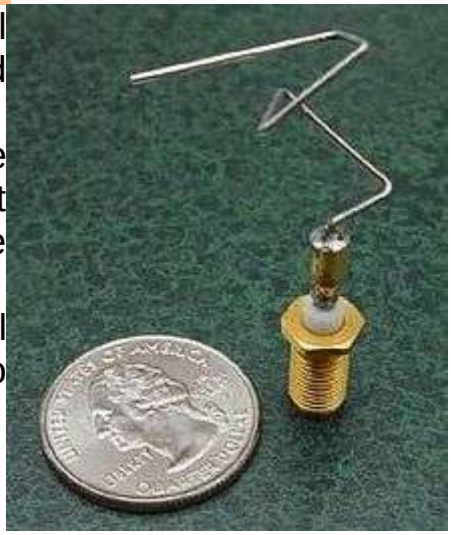
Objective

To be able

- to identify constants and variables in a simple problem
- to define the type of a constant
- to explain an algorithm
- to describe a simple problem using an algorithm

Why do we need algorithms ?

- Our world needs more and more technical solutions, more ergonomics and wants to find answers when new problems appear.
- Our powerful computers are able to manage billions of operations in a few seconds, but they need powerful softwares to organize the operations.
- To communicate and to explain things, to deal with difficulties and maintenance a team has to use description tools.

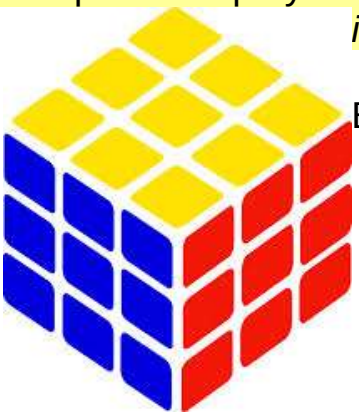


Algorithms help for all these issues.

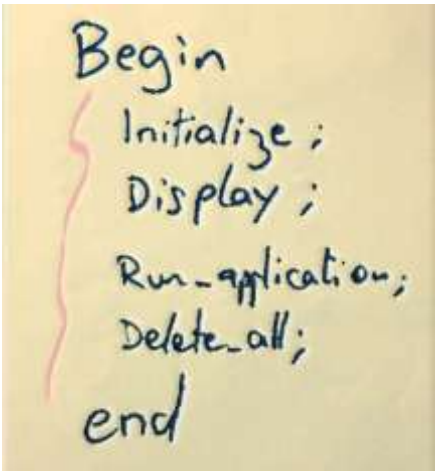
For example the picture below represents a complicated shape that was found by a designing program. It's an antenna that creates the best radiation pattern. A lot of engineers worked on this project and they needed powerful algorithms to produce this result.

What is an algorithm ?

An algorithm is a simple way to describe the resolution of a problem.
It explains step by step ***what to do*** to solve the problem , but not ***How to do*** *it*.



- Like a problem, an Algorithm always has a BEGINning and an END
- It's composed by several steps and each step solves a part of the problem.
- The more complex the problem, the more steps you need.
- Each task can be a smaller problem, with its own algorithm



What an algorithm is not...

- It's not a magic box. You have to understand the problem before you start the algorithm
- It's not a programming language. With a programming language you'll define how to solve the problem. But for some simple problems, algorithms are quite similar to a computing language.
- It's not a software. You don't need a computer and it's better to write it on a piece of paper
- It's not a gadget : it helps us to explain, simplify and solve complex questions.

Algorithm formalization

Algorithm composition

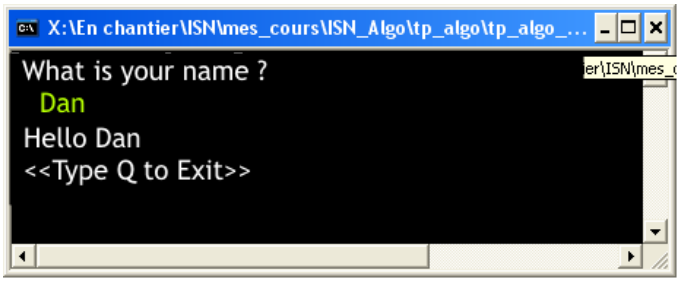
Linear algorithm

As we saw in a previous chapter, an algorithm :

- is the main description of a problem, or the description of a step from the main algorithm;
- starts with the keyword ***BEGIN***;
- ends with the keyword ***END***;
- is composed by simple words (often action verbs) which describe the operation.

Example

```
BEGIN
    prompt;
    get_name;
    write_hello;
    exit_app;
END
```



In this simple example :

- *prompt* displays the text "What is your name ?";
- *get_name* waits for the typing by the user;
- *write_hello* writes the text "Hello" on the screen, followed by the text written before;
- *exit_app* writes the text "<<Type Q to Exit>> on the screen and closes the application when the user types the letter Q.

Each of these steps could be a complex operation.
In this case, it's possible to make an algorithm for each of them. We'll talk about *sub-algorithm* in this case.

An Algorithm is also composed of *symbols* (constants and variables), of ***functions*** and ***procedures*** and of some standardized *pseudo-codes*.

Symbols

It's possible to define a symbol which is equivalent to a constant or variable value.

Constant symbol

A *constant* is a piece of information which never changes in an algorithm.
In the previous example, the text "What is your name?" is a constant string.
The number Π = 3,1416.... is a constant number.

Variable symbol

A *variable* is a piece of information which changes all along the algorithm or in an algorithm's life.
In the previous example, the name changed (one time *Dan*, another time *Sally*, etc...); it's a variable string.

Type of a constant or a variable

A constant or a variable actually is a piece of memory in the computer. It's necessary to define the number of memory cells needed by each variable or constant.
A byte needs more memory than a bit, a string more than a byte, etc....
Common types are :

- bit : 0 or 1
- byte : 8 bits from 0 to 254 (or -128 to +127)
- integer : from -32768 to 32767
- word : it's a double byte
- double (word) : 32 bits
- float (ing number) : 32bits to describe a rational number with a sign.
- char(acter) : it contains one letter
- string : it's a sequence of characters (most of the time with a maximum of 256)

The *get_name* and *write_hello* algorithm could be written like this :

var name:string; //name is a global string variable

```
Algorithm get_name
BEGIN
    read(name);
END
```

```
Algorithm write_hello
BEGIN
    write('Hello'+name);
END
```

Explanation

read is a standard command which affects to its argument the string typed by a user. The ENTER key defines the end of the word.
For example when the user types the keys on the keyboard:

D A N ENTER

the variable name is equal to DAN

write is a standard command which writes the argument text on the screen. This command displays the text between quotes. When there are no quotes, it should be a constant or a variable and in this case, this command displays its value.
For example, when name=Sally, the screen display will be Hello Sally.

Global and local symbols

- A *global* symbol is a symbol declared in the main Algorithm and is accessible in multiple scopes (in all the sub algorithm)
- A *local* symbol is declared within a sub algorithm and is not accessible in the other one.

In our example, the variable *name* must be accessible in *get_name* and in *write_hello*. It must be a global variable.

Procedure and Function

In a program's source code, the main program is the translation of the main algorithm.
The sub-algorithm could be a procedure or a function.

Procedure

A procedure is a part of a program (a sub-algorithm) that performs something : displaying a piece of information, saving a file, printing a text, etc....
In our example *prompt*, *get_name*, *wite_hello* are procedures.

Function

A function is also a part of a program that performs and calculates something. It returns a result to the caller.

Example :

```
function RandomVal :integer;
  begin
   // Get a random number from 1 to 3
    // Return this value as a int type in the return variable, Result
   Result := RandomRange(1, 3);
 end;
```

To use this function : ***Aleat:=RandomVal;***

Pseudo-code

Pseudo-codes are special commands for

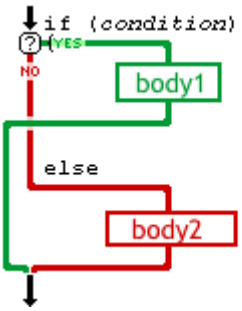
- conditional statement : IF.... THEN.. ELSE, CASE.... OF.
- Iterative statement : WHILE... DO, REPEAT...UNTIL

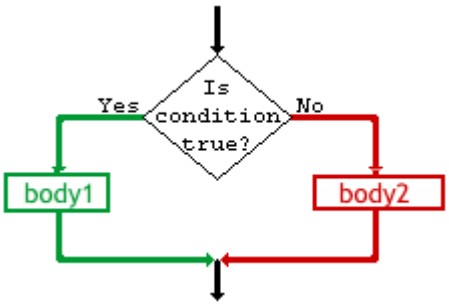
Conditional statement

A conditional statement is a rule which helps to decide what to do depending on a variable value.

There are two types of conditional statements :

If *(condition)* **then** *body1* **else** *body2* : If the condition of the operator 'if-else' is true, it passes the control to the first operator in ***body 1***. After all operators in body 1 have been executed, it passes control to the operator that follows the operator 'if-else'. If the condition of the operator 'if-else' is false, then:





- if there is the key word 'else' in the operator 'if-else', then it passes the control to the first operator in body 2. After all operators in body 2 have been executed, it passes control to the operator that follows the operator 'if-else';
- if there is no key word 'else' in the operator 'if-else', then it passes the control to the operator that follows the operator 'if-else'.

CASE *(variable)* **OF....** : The *CASE* operator provides a structured equivalent to a sequence of IF... THEN... ELSE statements on the same variable.

The CASE statement is more elegant, more efficient, and easier to maintain than multiple IF.. THEN... ELSE nestings.

Example :

```
CASE age OF
    0..3: write('Baby');
    4..14 : write('child');
    14..18 : write ('Teenager')
    19..150 : write('Adult');
END;
```

Comment : In some computing languages (C, PHP, java...) the CASE statement is replaced by a SWITCH statement.

Iterative statement

Iteration statements are used to specify the logic of a loop.

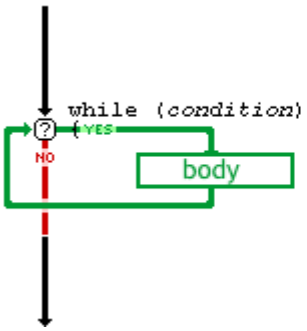
There are three kinds of rules :

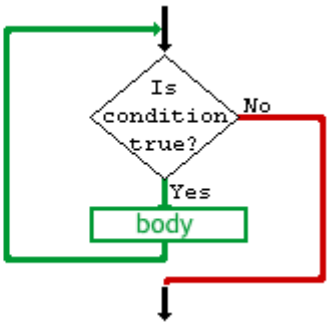
FOR *(condition of the loop)* **DO :** this statement is used when the number of loops is known.

Exemple :

```
FOR i:=0 TO 9 DO write(i+':');
```

The result will be 0:1:2:3:4:5:6:7:8:9:





WHILE *(condition)* **DO** : The WHILE keyword starts a control loop that is executed as long as the *condition* is satisfied. The loop is not executed at all if the expression is false at the start.

REPEAT... UNTIL *(condition)* : The REPEAT keyword starts a control loop that is always executed at least once, and which terminates when the *condition* is satisfied.

Example :

```
REPEAT
    write('Type Q to quit');
    read(c);
UNTIL (c='Q');
```

Break the loop when the Q character is typed on the keyboard.

Vocabulary

Algorithm	Algorithme
function	fonction
key	clé / touches dans le cas d'un clavier
keyboard	clavier
programming language	langage de programmation
a ***prompt***	une invite (de commande par exemple)
step by step	étape par étape
string (of letters)	chaîne de caractères