MI-GLSD-M1 -UEM213 : Programming languages paradigms

Chapter II: imperatif paradigm

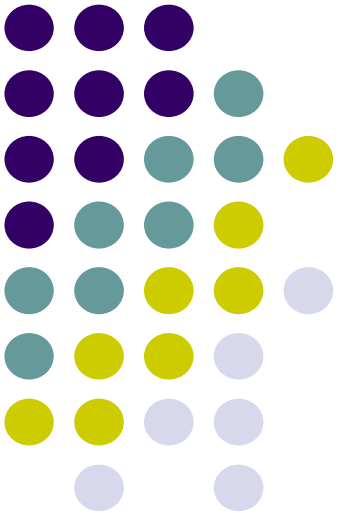
A. HARICHE

University of Djilali Bounaama, Khemis Meliana (UDBKM)

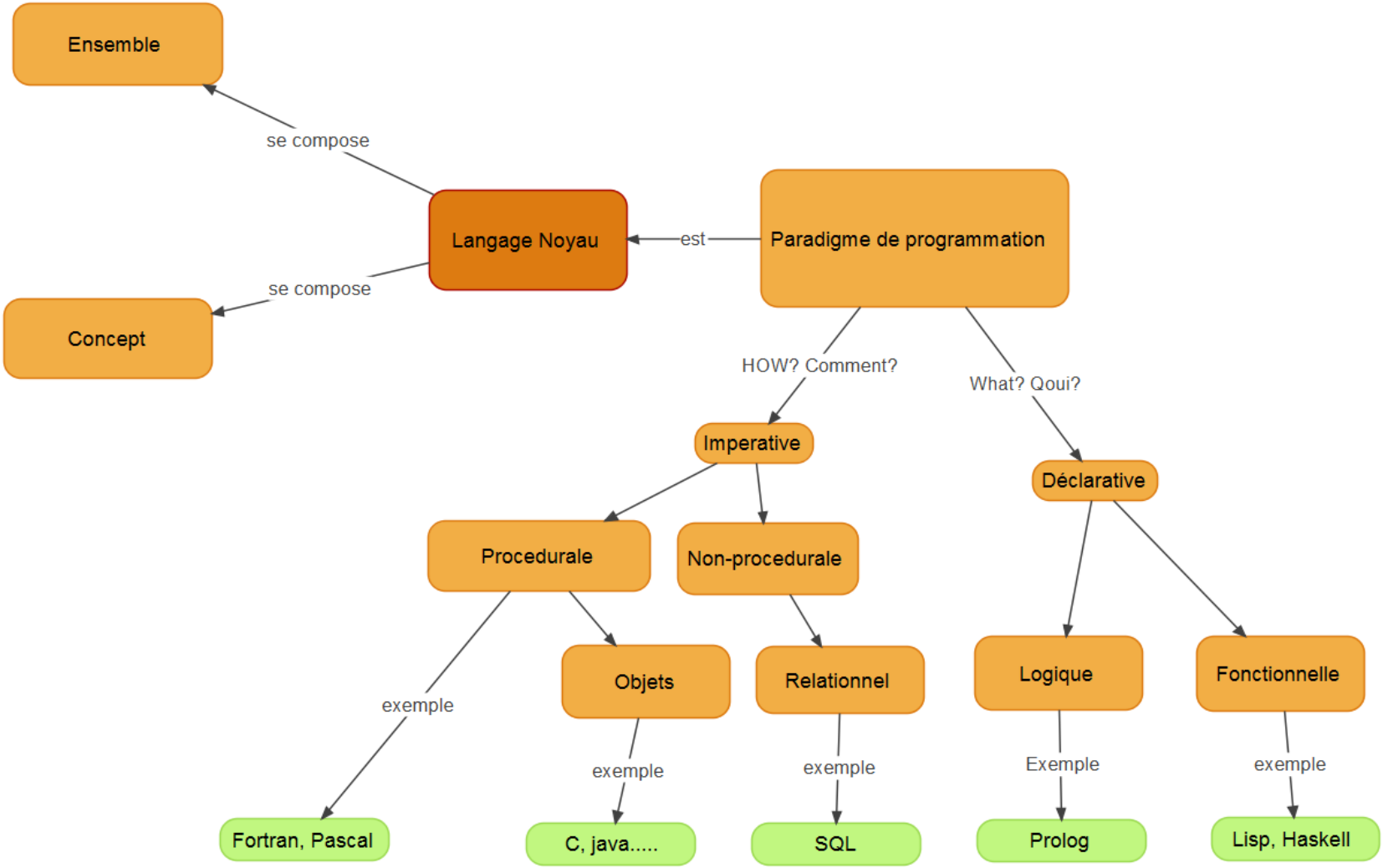
Faculty of Sciences & Technology

Mathematics & Computer Science Department

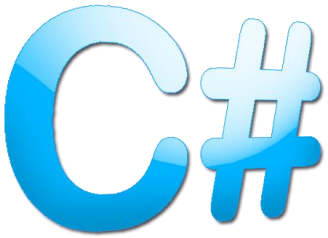
a.hariche@univ-dbkm.dz

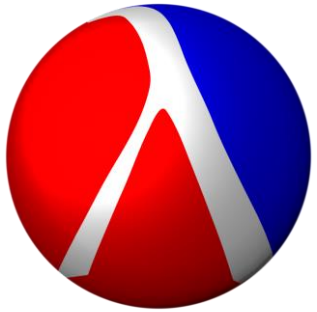


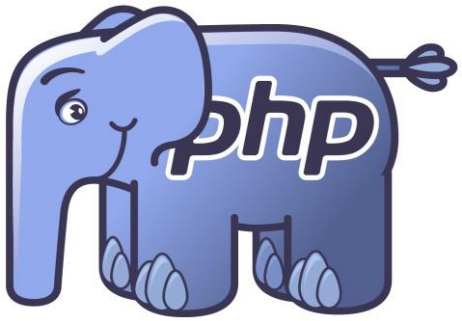
Reminder of the last lecture

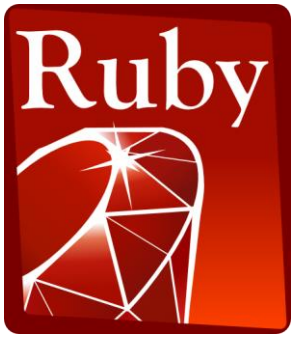


H u n d r e d s of programming languages are in use...

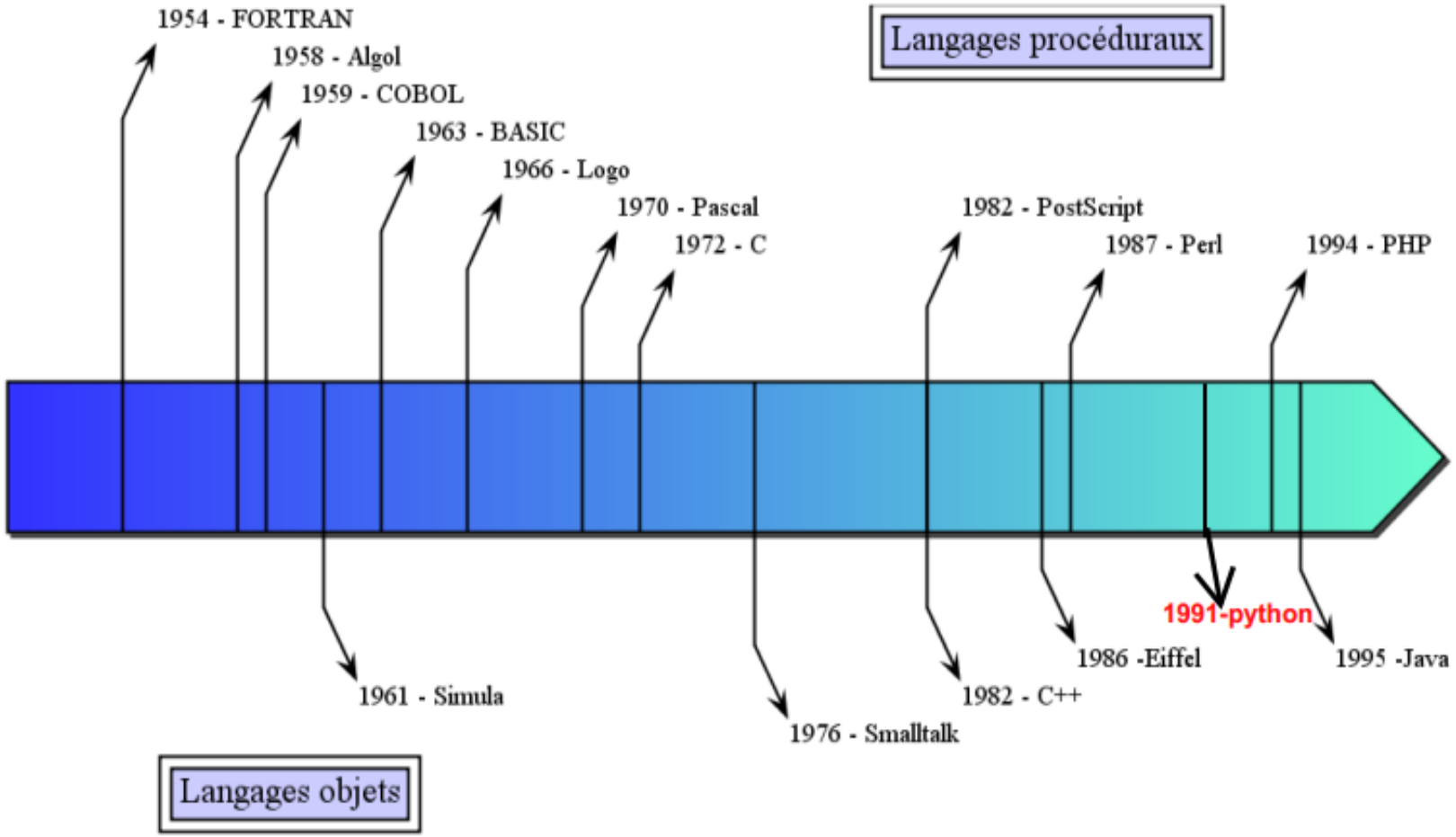




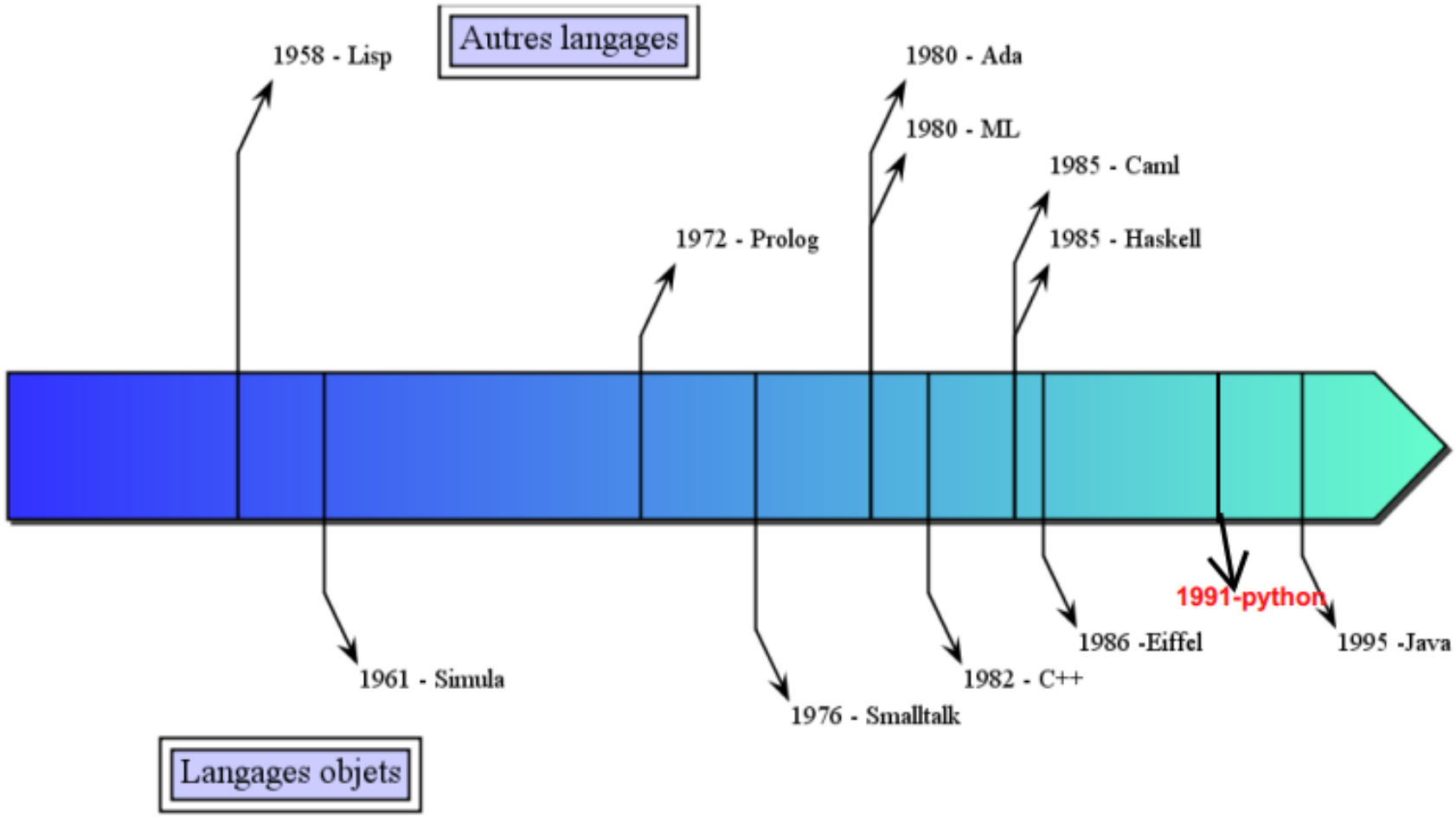




Timeline of programming languages



Timeline of programming languages



It’s the most useful paradigm why?

Advantages	flaws
Quite simple to read	Code becomes very large and loses clarity very quickly
The easiest to learn	Higher risk of errors when editing codes
Easy-to-understand in term of thought model for beginners (resolution by steps)	Re-coding operations block the development of applications, because the programming is closely linked to the system
Allows for the characteristics of special application cases	More difficulty for optimizations and extensions

Historical moment

Church (1930) et Turing (1936)

- Definable functions with lambda-calculus. (functional programming)

= Recusive function (functional programming)

= Turing machine (imperative programming)

- Effective calculus idea, formalization of computability
- Turing machine=realistic computer model.
- No Computer is yet build since !!
- Presence of non-computable functions.

Notions you may know

- Program
- Variable, type, declaring
- Conditional instruction
  - `If (…){……} else {…}`

- Iterations
  - `While (…){…}`
  - `Do {…} while(…)`
  - `For (…;….;…){…}`

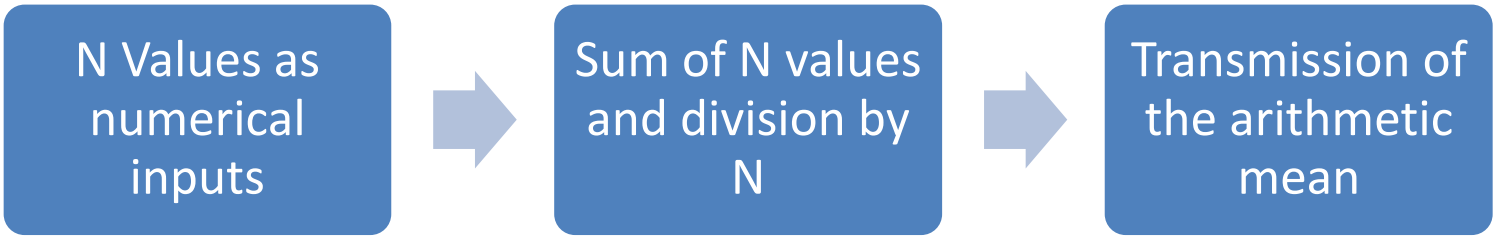
- Sub-program (procedure & function), calls of sub-program .

A.HARICHE a.hariche@univ-dbkm.dz

Data processing

Generally an application in computing sense means:

Example :



A.HARICHE a.hariche@univ-dbkm.dz

What is a program?

- Every processing launched by user for a machine and it is done by a sequence of operations called instructions i.e. a sequence of instructions create a program.

Remember
• A program is a sequence of instructions that makes a computational system to run a specific task

Written by a comprehensible programming language (directly or indirectly) by a computer.

An imperative program

Definition (Reminder)

An **imperative program** is a sequence of **instructions** that step-by-step specify actions to do aiming to get outputs from a set of inputs.

A.HARICHE a.hariche@univ-dbkm.dz

The foundations

- Imperative languages all have these basic instructions:
- **Assigning** (or assignment) : is used to store in memory (in a variable) the result of an operation.
- **Condition** : executes a block of instructions if a predetermined condition is met.
- **Loop** : allows a block of instructions to be repeated a predefined number of times or until a condition is met.
- **Branching:** Allows the execution sequence to be transferred elsewhere in the program (goto).
- **Sequence of instructions:** refers to the fact of executing, in sequence, several of the instructions under a given name (subroutines).

An imperative program Example

- Variables are modified by instructions.
- Variables are space memories with lablels.
- Instructions are sequencially organized.
- Machine is programmed by a side effects
  (= updating of general state machine (memory; video,…)

A space memory

```
Unsigned long int fact(unsigned int n){
Unsigned long int r=1UL;
For(;n>0u;--n)
r*=n;
return r;
}
```


A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Example

- Variables are modified by instructions.
- Variables are space memories with lablels.
- Instructions are sequencially organized.
- Machine is programmed by a side effects
  (= updating of general state machine (memory; video,…)

```
Unsigned long int fact(unsigned int n){
Unsigned long int r=1UL;
For(;n>0u;--n)
r*=n;
return r;
}
```

A space memory

Another

A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Example

- Variables are modified by instructions.
- Variables are space memories with lablels.
- Instructions are sequencially organized.
- Machine is programmed by a side effects
  (= updating of general state machine (memory; video,…)

```
Unsigned long int fact(unsigned int n){
Unsigned long int r=1UL;
For(;n>0u;--n)
r*=n;
return r;
}
```

A space memory

Another

r space is sequencially modified by a loop

A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Data & instructions

- A program is composed of instructions working on data
- Imperative programming data are saved inside variables.
- Hundred programming languages (C/C++/java…) insiste of **declaring** variables

```
#include<iostream>
void main(void) {
  double a,b;                    // Declarations
  a=1; b=1;
  while ((((a+1)-a)-1)==0) a*=2;
  while ((((a+b)-a)-b)!=0) b++;
  std::cout << "a=" << a << ", b=", b << std::endl;
}
```


A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Machine structure

Keep in mind

A machine is composed of 4 essential elements:

- **Memory** for saving data .
- **Calculus** unit (logic, arithmetic…)
- **Command** unit that manage all the work
- **Communication** units (keyboard, screen, net-interfaces,….)

A.HARICHE a.hariche@univ-dbkm.dz

An imperative program
The memory

The main memory of a computer is composed of :

- Tremendous number of **capacitors (10^{10}-10^{11})** could be powered or unpowered with (0 or 1)
- a gigantic network of wires and switches

Keep in mind

The capacitors are in-grouped by (typically 8, 16, 32 or 64) called **words**. a word can be located in memory thanks to a number called **address.**

The operator & returns the address of a variable:

```
1    int a = 5;
2    cout << a << " " << &a << endl;
```

This will show something like : `5 0x7ffe94f3a8bc`

A.HARICHE a.hariche@univ-dbkm.dz

An imperative program
The variable

- Goal : save data in central memory during the execution of a program
- Avoid managing directly addresses and manage **variables** instead.
- Programmer gives variables names with his own choice (**identifiers**)

Definition -Variable

A **variable** is a space memory where a program can memorise data.

Variables links one or multiple memory cells containing a sequence of 0 and 1.

A.HARICHE a.hariche@univ-dbkm.dz

An imperative program
The variable

1.Name

- Identify and refer into a variable
- Make link between the problem and the programming language

2.Location

- Locate by an address inside an entity
- Address created at the moment of variable is created

3.Value

- The content of a cell memory linked into a variable
- The type help to extract and interpret the content

4.Scope

- The extension of program where the variable is known
- Set of locations referring into an existing variable
- A visible variable in its scope is invisible in other-else.

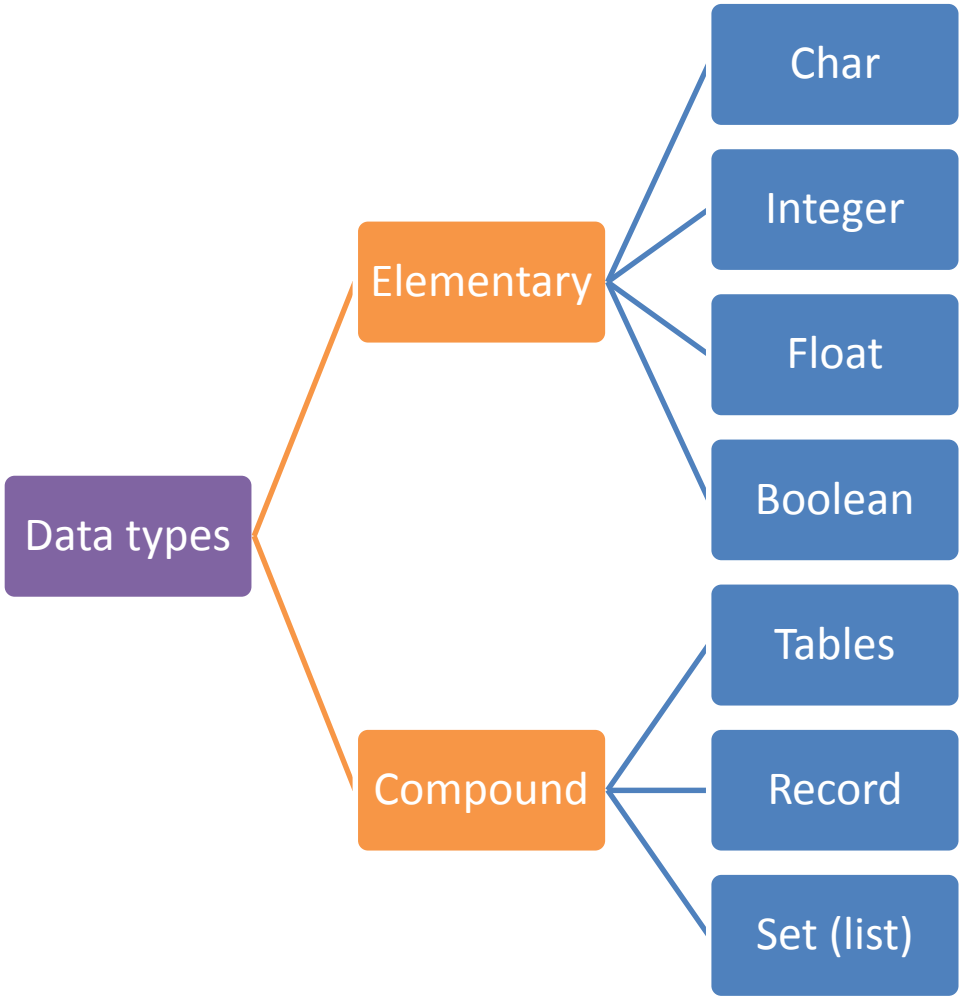
5.Life-time

- Time period where a memory location of a variable still available

6.Type

- An abstraction defines the set of values and all possible operations for a such a variable

An imperative program
The variable



An imperative program Declaring a variable

- A variable can be linked into a physical location during:

- compilation (rarely),
- uploading (static allocation),
- execution (dynamic allocation);

- In implementational view point, the declaration of a variable is used to determine the amount of memory space required by the program ;

```
PASCAl: var name : type;
C: Type name;
```


A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Assignment

- variable and its value are linked by an assignment:

Example : V:= E:

«assign the name V to the value of the expression E until the name V is reassigned to another value»

PASCAL: V:=E
C: V= E;

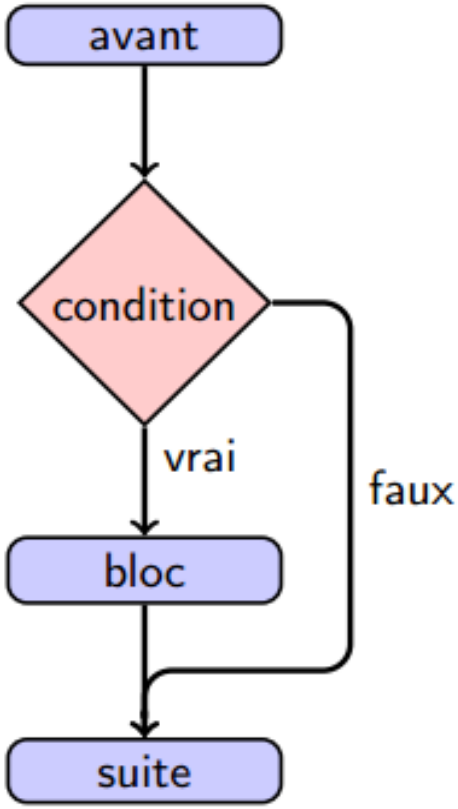
- Assignment is not a constant definition:

```
X:=3;
X:=X+1;
```


A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Conditional

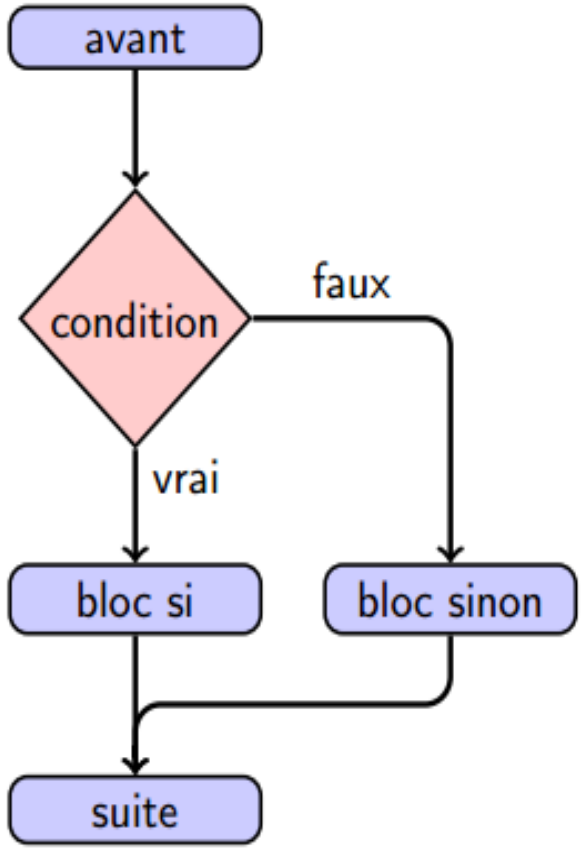
```
Instructions before;
If(condition){
   instructions block;
}
Instructions after;
```



Remark: Braces are not used if there's only one instruction inside the block; choose the most **readable** way to code.

An imperative program Conditional

```
before;
If(condition){
    if block;
}else{
    else block;
}
 after;
```



A.HARICHE a.hariche@univ-dbkm.dz

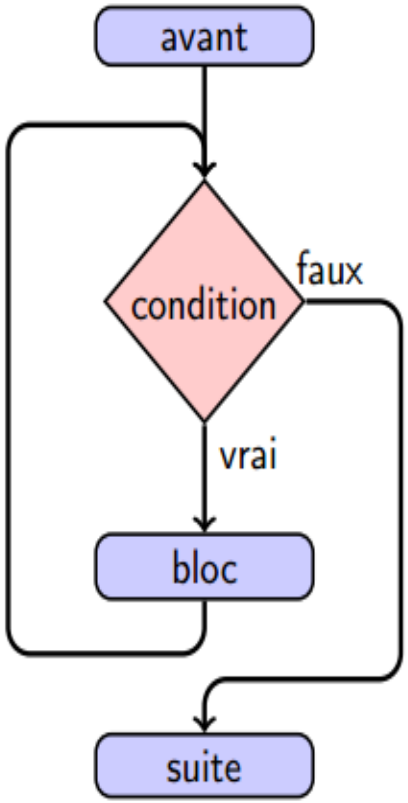
An imperative program Iteration(WHILE-Loop)

```
Instructions before;

while(condition){

      instructions block;
}

Instructions after;
```



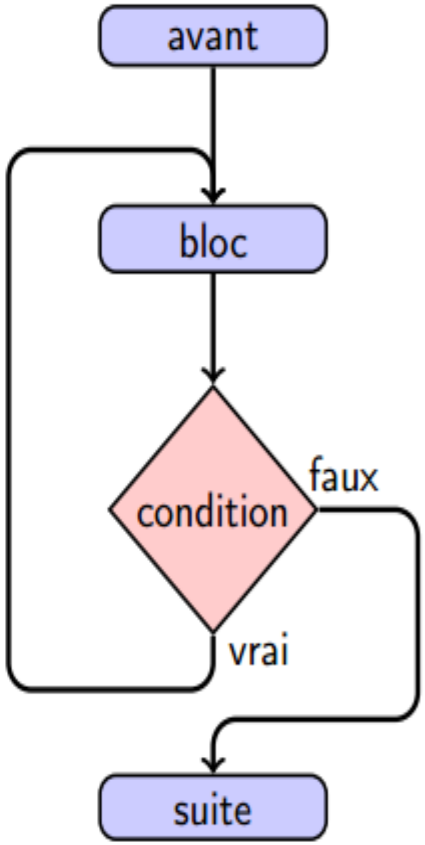

```
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5    int n;
6
7    cout << "Saisir un nombre positif :  ";
8    cin >> n;
9    while (n < 0) {
10       cout << "Ce nombre est négatif !" << endl;
11       cout << "Saisir un nombre positif :  ";
12       cin >> n;
13   }
14   cout << "le nombre saisi est " << n << endl;
15   return EXIT_SUCCESS;
16 }
```

Remark: if condition is false from the beginning the block will **never** be executed.

An imperative program Iteration(Do-WHILE-Loop)

```
Instructions before;
Do{
      instructions block;
}while(condition){
}

Instructions after;
```




```
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5    int n;
6
7    do {
8      cout << "Saisir un nombre positif :  ";
9      cin >> n;
10     if (n < 0) {
11         cout << "Ce nombre est négatif !" << endl;
12     }
13   } while (n < 0);
14   cout << "le nombre saisi est " << n << endl;
15   return EXIT_SUCCESS;
16 }
```

Remark: the block can **always** be running once at least.

A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Iteration(For-Loop)

Syntax

```
For (initialization; condition; incrementation){
Instruction;
}
```

It's **the most readable version** of a while loop:

```
initialization;
while(condition){
     instruction;
     incrementation;
}
```


A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Sub-program

Syntax

Declaring (functional mode):

```
return_type name(type1 param1,type2 param2 ….)
```

Defintion:

```
return_type name(type1 param1,type2 param2 ….){
Declaring locale variables;
Instructions;
…
return return_value;
}
```

Call (uses-case inside an expression):

```
… name(type1 param1,type2 param2 ….)…
```


A.HARICHE a.hariche@univ-dbkm.dz

An imperative program
Sub-program –example-

Syntax

```
int somme(int, int);

int somme(int a, int b) {
  int res;
  res = a + b;
  return res;
}
```

```
int main() {
  int x, y, s;
  cin >> x >> y;
  s = somme (x, y);
  cout << s << endl;
  return EXIT_SUCCESS;
}
```


A.HARICHE a.hariche@univ-dbkm.dz

An imperative programSub-program –return command-

Keep in mind

Two roles:

- **Quit** the current function;
- **return** a value into the calling program.

- In a void only the first one (the return value is null) is used.
- The return is implicitly existing always inside a void.

A.HARICHE a.hariche@univ-dbkm.dz

An imperative programSub-program –return command-

Keep in mind

Two roles:

- **Quit** the current function;
- **return** a value into the calling program.

- In a void only the first one (the return value is null) is used.
- The return is implicitly existing always inside a void.

A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Sub-program –Recursion-

- Exercise: Write using a sub-program n!

A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Sub-program –Recursion-

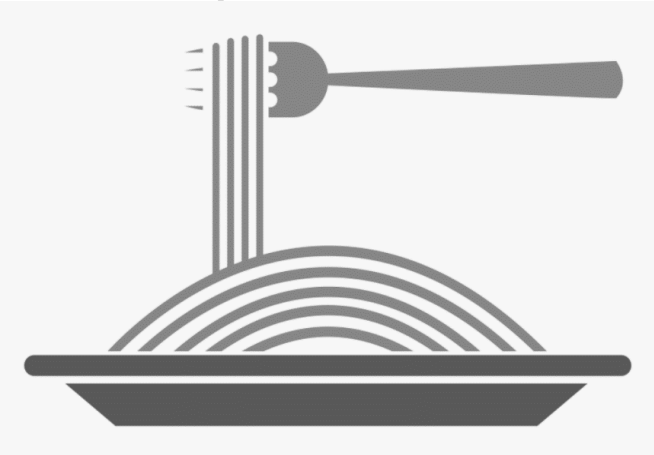
- Exercise: Write using a sub-program n!
- Solution:

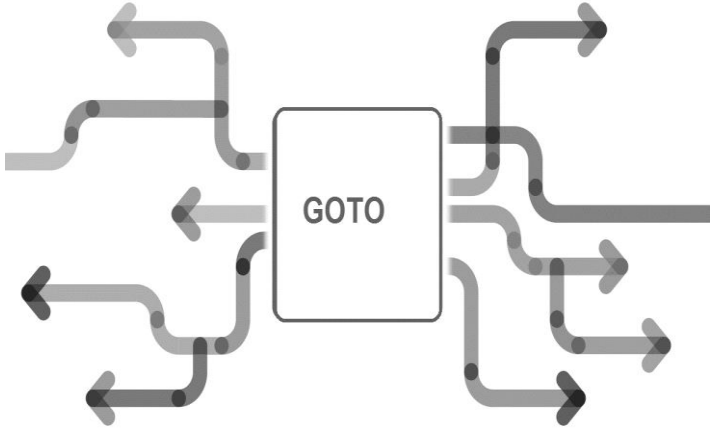
```
int factorielle(int n) {
    if ( n == 0 ) {
        return 1;
    } else {
        return n * factorielle(n-1);
    }
}
```


A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Branching

- The GOTO command is the most powerful, and the most critical mainly for two reasons :
- except in specific cases, it is possible to perform the same action more clearly using control structures
- Using this statement can cause your code to be harder to read and, in the worst cases, it makes spaghetti code (problem case on software system of Toyota camry 2005 in 2007).

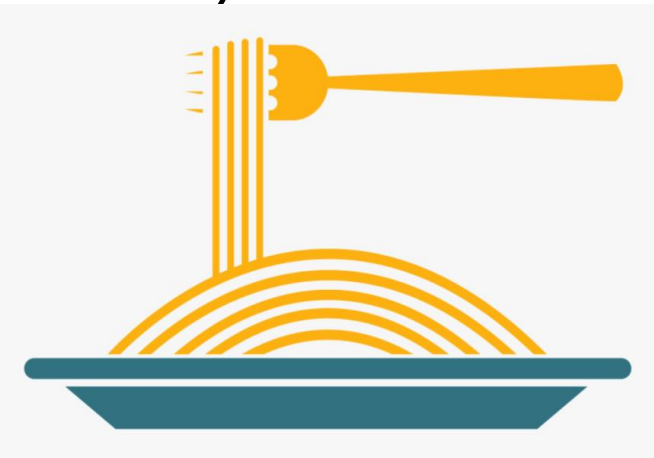




A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Branching

- The GOTO command is the most powerful, and the most critical mainly for two reasons :
- except in specific cases, it is possible to perform the same action more clearly using control structures
- Using this statement can cause your code to be harder to read and, in the worst cases, it makes spaghetti code (problem case on software system of Toyota camry 2005 in 2007).



A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Branching

- The GOTO command is the most powerful, and the most critical mainly for two reasons :
- except in specific cases, it is possible to perform the same action more clearly using control structures
- Using this statement can cause your code to be harder to read and, in the worst cases, it makes spaghetti code (problem case on software system of Toyota camry 2005 in 2007).

In fact, it is not recommended to use it.

A.HARICHE a.hariche@univ-dbkm.dz

An imperative program Branching

- Exercise : Here is a spaghetti code problem

Get a solution using iterations .

```
i = 0;
 twenty: i ++;
If (i != 11) { GOTO eighty;}
 if(i = 11) { GOTO sixty;}
 GOTO twenty;
 eighty: printf(" %d”,i ," au carré = %d\n“, i * i);
GOTO twenty;
sixty:printf(" Programme terminé.“);
```

An imperative program Branching

- Solution : a for loop can solve the problem.

```
for (i = 1; i <= 10; i++)  printf(" %d”,i ," au carré = %d\n“, i * i);
printf(" Programme terminé.“);
```


A.HARICHE a.hariche@univ-dbkm.dz

Imperative Paradigm
Difficult reasoning

- Some axioms are valid **only if the language does not include synonymy and side effects**;

- Language theory works attempt to explain Programming languages in terms of **well-defined constructs**;

- **This complexity of reasoning** was a strong motivation to provide solutions like **functional and logical paradigms**.

A.HARICHE a.hariche@univ-dbkm.dz

Imperative Paradigm Escapement

- It is a command that terminates the execution of a textually enclosing construct:
  - Return Exp,

is used in C to exit a function call and return the value calculated by the function.

  - Exit en ADA/ Break en C,

allows program control to be transferred from where the exit command is located to the first command after the innermost loop following the exit

  - Continue en C,

transfers control to the beginning of the enclosing loop.

A.HARICHE a.hariche@univ-dbkm.dz

Imperative paradigm
Exceptions

- This is an "abnormal" event that occurs during execution

Exercise: how to make a program calculating the function $\sqrt{x}$ in C to solve the following exception of the function *pow()* :

The call is equivalent to requesting the result of the expression $-1^{\frac{1}{2}}$, in other words, from this expression: $\sqrt{-1}$, which is impossible in the set of reals.

A.HARICHE a.hariche@univ-dbkm.dz

Imperative paradigm
Exceptions

Solution: by using *errno* library:

```
#include <errno.h>
#include <math.h>
#include <stdio.h>


int main(void)
{
   errno = 0;
   double x = pow(-1, 0.5);

   if (errno == 0)
      printf("x = %f\n", x);

   return 0;
}
```

Where *errno* can have the values:
EDOM (for the case where the result of a mathematical function is impossible),
ERANGE (in case of overflow, we will come back to this later)
EILSEQ (for conversion errors, we'll talk about that later as well).
-These two constants are defined in the header <errno.h>.

A.HARICHE a.hariche@univ-dbkm.dz

Imperative paradigm
Side effects

- Is used to enable communication between program units in a way that:
  - The function do an I/O updating (1)
  - The global variable is updated (2)
  - The local permanent variable is modified (3)
  - A pass-by-reference parameter is modified (4)
  - A change is made to non-local/static variables via pointers (5)
  - Any called function also satisfies one of these conditions (6)

Imperative paradigm
Side effects

Exercise: Investigate the side effects of this program that contain f1, f2, f3:

```
program demo;
var b : integer;
function f3 : integer;
var x : integer;
begin
x := f1 (x);
x := f2
end;
function f2 : integer;
var x : integer;
begin
x : = 20;
f2 := x
end;
```

```
function f1 (var a : integer) : integer;
begin
read (a);
a := f2;
a := f3
end;
begin
write (f1(b));
end.
```


A.HARICHE a.hariche@univ-dbkm.dz

Imperative paradigm
Side effects

Solution: f1 has a side effects since the conditions 1 & 4 are checked.

Function	Nonlocal	Ref	I/O	Indirect	Side effect
f1	Ø	[a]	[R]	Ø	true
f2	Ø	Ø	Ø	Ø	?
f3	Ø	Ø	Ø	Ø	?

Imperative paradigm
Synonymie (Aliasing)

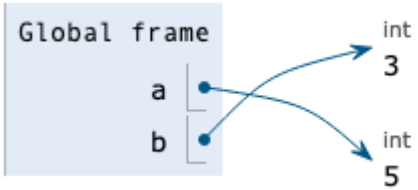
- Two names are aliases if they denote (share) the same data object during a unit launch.
- Synonymy is another feature of the imperative programming language that makes programs harder to understand.

Example 1

```
procedure confuse (var m, n : Integer );
 begin
 n := 1; n := m + n
 end;
 ...
 i := 5;
 confuse(i,i)
```

Example 2

```
a = 3
b = a
a = a + 2
```



m & n are bound to the same variable i, initialized to 5; after the call the value of i is 2 and not 6.

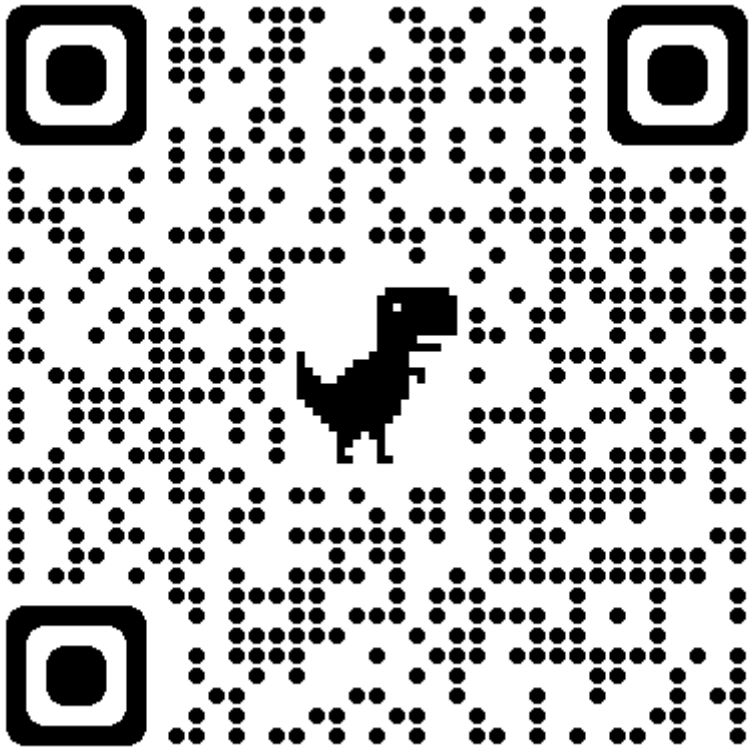
A.HARICHE a.hariche@univ-dbkm.dz

Imperative Paradigm Briefly

- Imperative programming is based on the Von Neumann machine model.
- In the imperative paradigm is characterized by programming with state and commands that modify the state.
- The difficulties associated with imperative paradigms can be summarized as follows:
  - Side effects
  - Synonymies (aliasing)
  - Exceptions

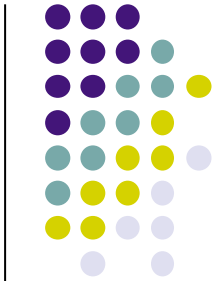
- **The complexity of reasoning** stemming from the difficulties of imperative programming has been a strong motivation to provide solutions such as **functional and logical paradigms**.

PLP_Drive space



https://drive.google.com/drive/folders/1YBCIZzAldeiT19DIfDiREQwP-NAQ1qMN

Many important ideas



Louv1.1x

- Identifiers and environments
- Functional programming
- Recursion
- Invariant programming
- Lists, trees, and records
- Symbolic programming
- Instantiation
- Genericity
- Higher-order programming
- Complexity and Big-O notation
- Moore's Law
- NP and NP-complete problems
- Kernel languages
- Abstract machines
- Mathematical semantics

Louv1.2x

- Explicit state
- Data abstraction
- Abstract data types and objects
- Polymorphism
- Inheritance
- Multiple inheritance
- Object-oriented programming
- Exception handling
- Concurrency
- Nondeterminism
- Scheduling and fairness
- Dataflow synchronization
- Deterministic dataflow
- Agents and streams
- Multi-agent programming