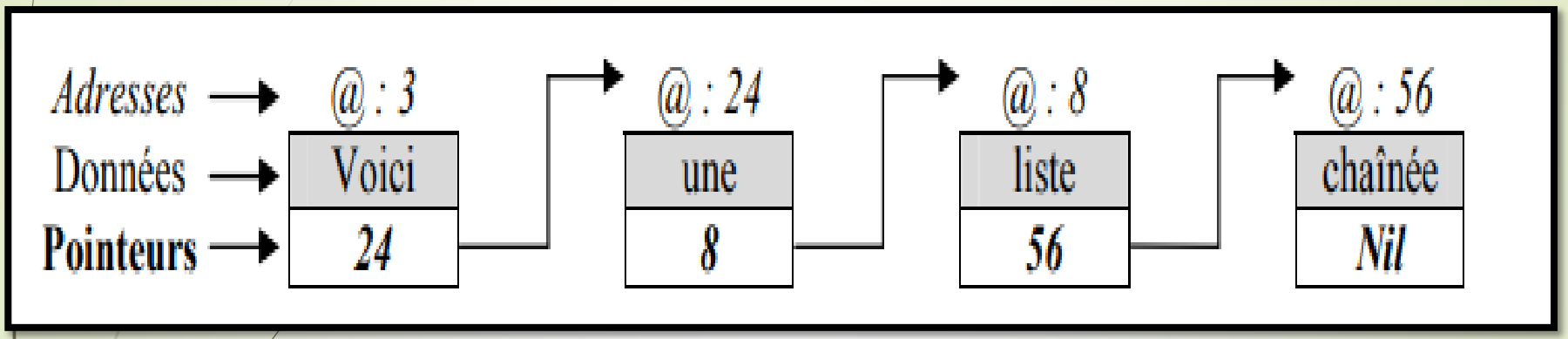
Linked lists

ASD2

Definition

- It is a linear structure which **does not have a fixed dimension** at its creation.
- Its **elements of the same type** are **scattered** in memory and **linked** together by **pointers** .
- **Its size** can be **modified** depending on the space available in memory.
- The **list is accessible only** by its **head** of list, that is to say its **first element** .

Illustrative example of a list



- This example shows a **linked list** of strings.
- To access the **third element** of the list you must always **start reading** the list with its **first element** in the **pointer of which the position of the second** element is indicated. In the **pointer of the second** element of the list **we find the position of the third element** ...

Illustrative example of a list

- **The sequence** of a linked lists is implemented by the **pointer carried by each element** which **indicates the location** of the **next element** .
- The **last element** of the list **points to nothing** (**Nil**).
- We access an element of the list by traversing the elements using their **pointers** .

Types of linked lists

There are different **types** of linked lists:

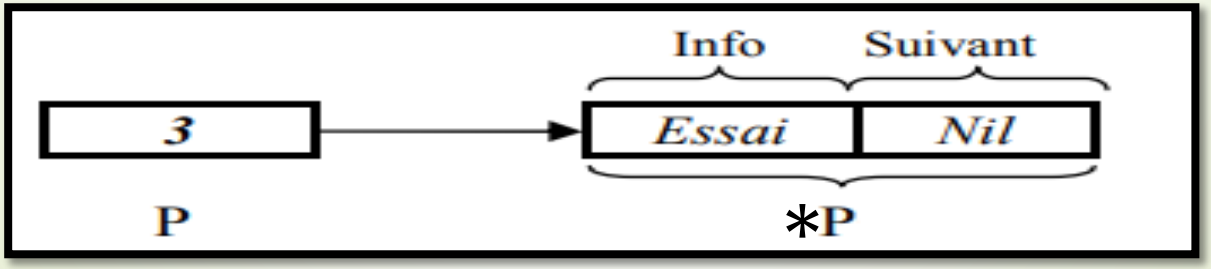
- ***Simple linked list*** consisting of elements linked together by pointers.
- ***Ordered linked list*** where the next element is larger than the previous one. Element insertion and deletion are done so that the list remains sorted.
- ***Doubly linked list*** where each element has two pointers pointing respectively to **the previous element** and **the next element** . This allows the list to be read in both directions, from the first to the last element or vice versa.
- ***Circular list*** where the last element points to the first element in the list. If it is a doubly linked list then the first element also points to the last.

Linked List Vs Array

Structure	Dimension	Position d'une information	Accès à une information
Tableau	Fixe	Par son indice	Directement par l'indice
Liste chaînée	Evolue selon les actions	Par son adresse	Séquentiellement par le pointeur de chaque élément

Linked lists

- An element of a list is a structure formed:
  - of data or information,
  - a pointer named **Next** indicating the position of the next element in the list.



Linked lists

The pointer variable **P** points to the memory space **(*P)** of address **3** .

This memory cell contains the value " **Essai**" in the **Info field** and the special value **Nil** in the **Next field** . The **Next field** will be used to indicate the position of the next element which represents a part of a list.

The value **Nil** indicates that there is no next element.

*P is the object whose address is stored in P.

Linked lists

- Each element is associated with a **memory address.**
- Linked lists use the **allocation** and **Free function**. **Allocate(P)** and **Free(P).**

Simple linked list

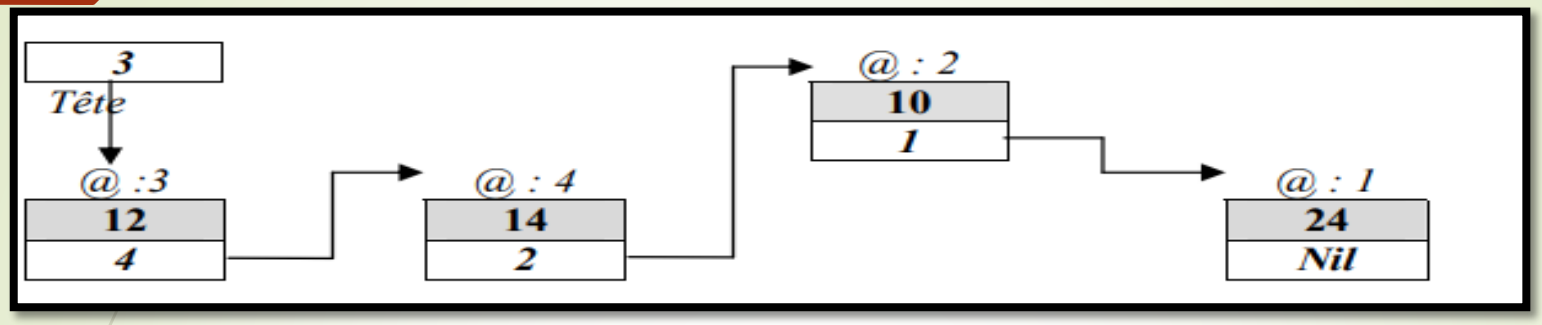
Simple linked list is composed:

- of a **set of elements** having the previous structure.
- of a variable, called **Head** , containing the address of the first element of the linked list.
- The last element pointer contains the value Nil .

In the case of an **empty list** the head pointer **contains the value Nil** .

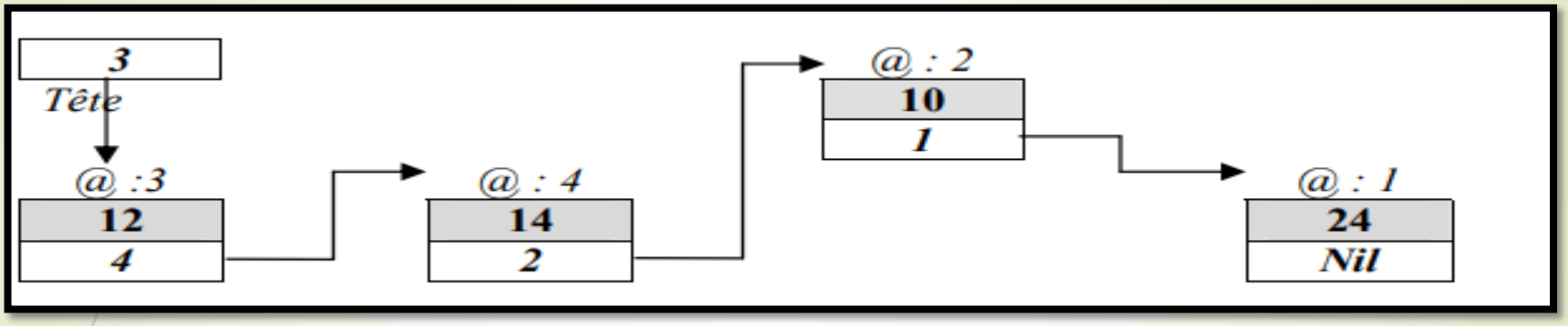
A **list is defined** by **the address of its first element** .

Example of a list of integers



- The 1st element of the list is 12, stored at address 3 (start of the linked list).
- The 2nd element of the list is 14, stored at address 4 (because the pointer to cell at address 3 is equal to 4).
- The 3rd element of the list is 10, stored at address 2 (because the pointer to cell at address 4 is equal to 2).
- The 4th element of the list is 24, stored at address 1 (because the pointer of cell at address 2 is equal to 1).

Example of a list of integers



- If P has the value 3:
  - **(*P).Info** has the value 12.
  - **(*P).Next** has the value 4.
- If P has the value 2:
  - **(*P).Info** has the value 10.
  - **(*) P.Next** has the value 1.

List type definition

Type Element = Record

Info: data_type ;

Next: *** Element** ;

EndRecord

Type List : * Element ;

The **type of Info** depends on the values contained in the list: **integer** , **real** , **character, string** , etc.

List in C language

```
typedef  struct  element
{
    int  info ;
    element * next;
} element ;
typedef  struct element * List;
List head, P;
```

Basic List Operations

- The treatments on the lists are as follows:
  - Create a list.
  - Browse a list.
  - Search for a value in a list.
  - Add an item.
  - Delete an item.
  - Edit an item.

Basic List Operations

- In everything that follows, we use a linked list of integers, its definition is given by:

Type Element = Record

info: int;

next: * Element ;

EndRecord

Type List : * **Element** ;

Creating a simple linked list

- We distinguish two cases:

a) The **number** of elements to create is **known** .

b) The **number** of elements to create is **unknown** .

Creating a simple linked list (known number of elements)


```
Algorithm CreationListNumberKnown ;
Type Element = Record
 info : integer;
 next : * Element ;
EndRecord
Type List: * Element ;
Variable head, P: List; N, i: int;
Begin
 head ← Nil;
Write ("enter the number of elements in the list"); Read (N);
 For i←1 to N do
  Allocate (P) ; Write (“Enter element value”);  Read ((*P).info);
  (*P).next ← head ;  head ←P;
 EndFor
END .
```

Creating a simple linked list (unknown number of elements)


```
Algorithm CreationListNumberUnknown ;
    Type Element = Record
     info : int;
     next : * Element ;
    EndRecord
    Type List: * Element ;
Variable head, P: List; Bool : Boolean; char: character;
Begin
 head ← Nil; Bool ← True;
 While ( Bool = true ) do
  Allocate (P) ; Write ("Enter element value"); Read ((*P).info); (*P).next ← head ; head ←P;
  Write (" Are there elements: Y/N" ); Read (char);
            If (char = "N") Then
                        Bool ← False;  End if
 EndWhile
END .
```

Displaying elements of a linked list

- ***A simple linked list can only be traversed from the first to the last element***

```
Algorithm ShowList ;
Type Element = Record
        info: integer ;
        next: * Element ;
EndRecord
       List: * Element ;
Variable head, P: List ;
Begin
 P ←head ; //point to the first element.
 while (P < > Nil ) Do // Scan the list from the first element to the last element.
  Write ((*P).info); // display element
   P← (*P).next; // move on to the next element
 EndWhile
END .
```

Find a given value in a linked list


```
Algorithm FindListValue ;
    Type Element = Record
        info : integer;
        next : * Element ;
    EndRecord
    List: * Element ;
Variable head, P: List; Bool : Boolean; val: integer;
Begin
Write("Enter the desired value"); Read(val);
If ( head <> Nil) Then
P ← Head ; Bool ← False;
 while ( Bool = False) and ( P < > Nil) do
If ((*P).info = val) Then
        Bool ← true;
 Else
        P ← ( *P).next ;
 End if
EndWhile
```

Find a given value in a linked list


```
If ( Bool = true) Then
    Write ("the value", val, "is in the list");
else
   Write( ("the value", val, "is not in the list");
End if
Else
  Write ( "the list is empty");
End if
END .
```

Inserting an element into a linked list

- 1) Insert at the beginning of the list

```
Algorithm InsertStartList
Type Element = Record
        info : integer;
        next : * Element ;
EndRecord
List: * Element ;
Variables head , P: List; val: integer;
Begin
Allocate(P);
 Write ("enter the value to insert"); Read (val);
(*P).info ←val;
(*P).next ← head ;
head ←P;
END .
```

Inserting an element into a linked list

- 2) **Insert at the end of the list**

Algorithm InsertEndList

Type Element **= Record**

info **: integer;**

next **: * Element ;**

EndRecord

List: * Element ;

Variable head, P,k : List;

Begin

k ← head ;

Allocate(P); Write ("enter the value to insert");

Inserting an element into a linked list

- 2) **Insert at the end of the list**

```
Read ((*P).info); (*P).next ← Nil;
If ( head = Nil) Then
  head ←P;
else
while ( (*k).next <> Nil ) Do
k ← (*k).next;
 endwhile
(*k).next ←P;
End if
END
```