Departamento de Informática
Universidad de Valladolid
Campus de Segovia

CHAPTER 2: POINTERS AND DYNAMIC MEMORY ALLOCATION

POINTERS AND DYNAMIC MEMORY ALLOCATION

- Introduction
- Pointers. declarations in Pascal
- Dynamic memory allocation.
- Basic dynamic variables operations
- Basic pointer operations
- The NIL value
- Some non recursive applications using pointers

INTRODUCTION

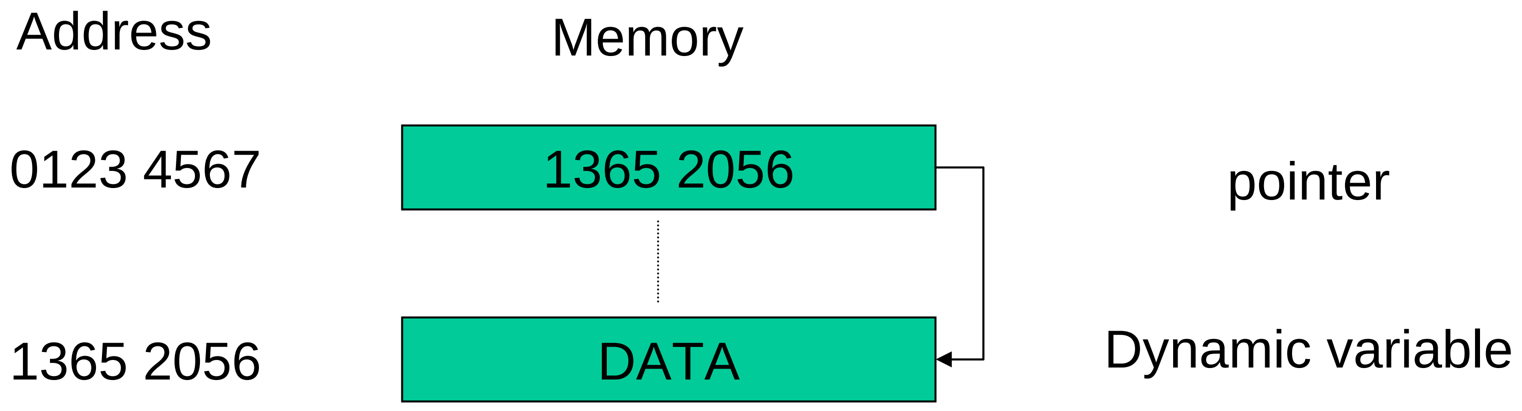
- All Data structures that have been presented until now can be considered as static since its size and existence are determined at compile time. It means:
  - The memory space allocation is reserved in advance and not change during program execution.
  - It allows the compiler to check the data types at compile time.
- Disadvantages:
  - Since the size of the static data structure doesn´t change during program execution then they aren´t suitable to optimize the spatial complexity.

INTRODUCTION

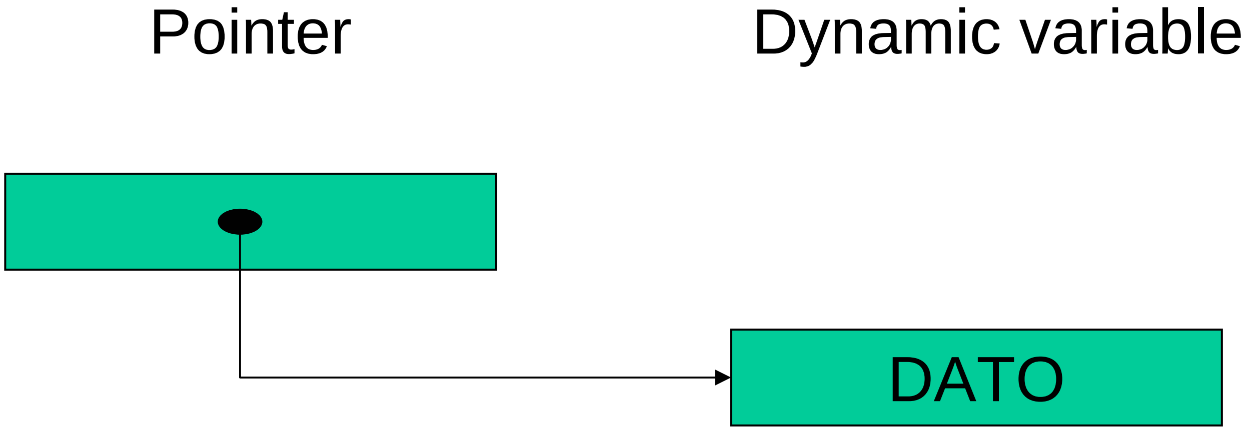
- To allocate space dynamically in Pascal is necessary to declare a pointer-type variable.
- Advantages:
  - Flexibility with respect to the data structures that can be implemented (list, trees, graphs,…)
- Disadvantages
  - **Alliasing**: As a collateral effect when the same memory space is allocated to two variables (two differents identifiers)
  - **Space Memory management**: Since is necessary to know at run-time how many memory is available and how many can be recovered if it is not used at this moment (run out of problems).

INTRODUCTION

- A **pointer** in pascal is a type of data that can only contain the addresses in memory of stored data (typed pointers).
- The **allocated memory space** is represented by a dynamic variable whose address in memory is contained by the pointer.



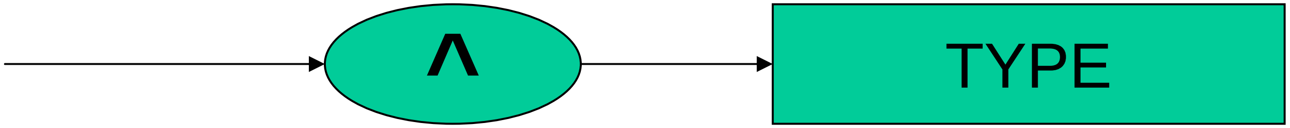
GRAPHICAL REPRESENTATION



HOW TO DECLARE A POINTER IN PASCAL

- To use a pointer in Pascal :
  - First, it is necessary to declare, in the TYPE section, the **pointer type**, it means, the data type that will be pointed by the pointer.
  - And second, in the VAR section, declare the pointer-type variables.

SYNTACTIC DIAGRAM



EXAMPLE I:

```
TYPE
        tpchar=^char;
VAR
        pchar:=tpchar;
```

EXAMPLE II:

```
TYPE
        tpNode=^tNode;
        tNode=record
                info:.......
                Sig:tpNode
        end;
VAR
        pNode:=tpNode;
```

Some aspect to take into account

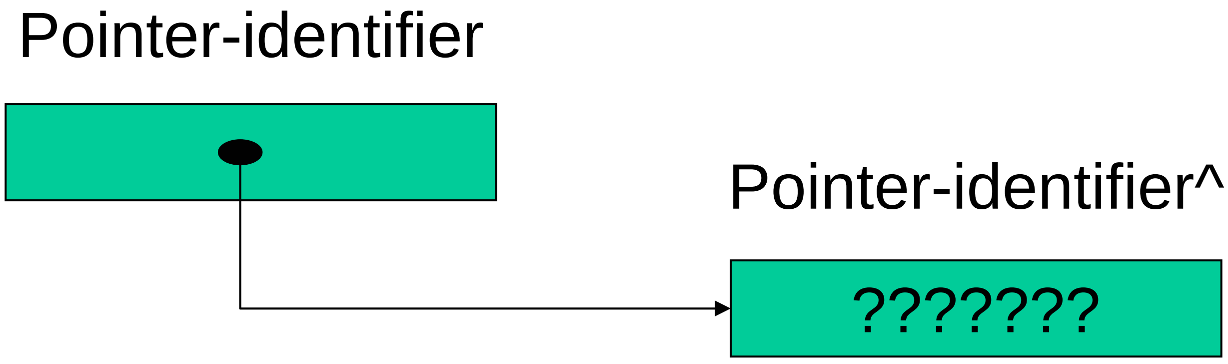
- The pointer size is independent of the pointer type.
- The dynamic variable no use memory space at compile-time but just when they are created at run-time.

CREATION AND DELETION OF DYNAMIC VARIABLES

- Pascal provide the next procedures to create or delete dynamic variables:
  - **New(pointer-identifier)**
  - **Dispose(pointer-identifier)**

CREATING A DYNAMIC VARIABLE IN PASCAL

- New(pointer-identifier)
  - A memory space whose size is related to the pointer type is used.
  - Then, its address is assigned to the pointer.
  - The new variable is denoted as:
    pointer-identifier^
- Graphically:



DELETION OF A DYNAMIC VARIABLE IN PASCAL

- Dispose(pointer-identifier)
  - Release dynamically allocated space.
  - What happens when Dispose procedure is called? It depends on the compiler version.
    - In some cases the pointer is set to NIL value,
    - In others is left unchanged with what looks like a valid address stored in it.

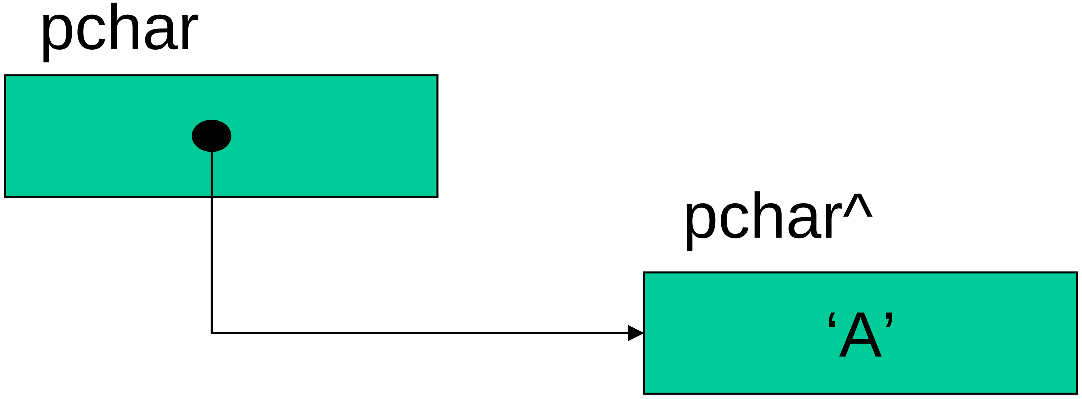
BASIC DYNAMIC VARIABLE OPERATIONS

- The allowed operations are
  - allocation
  - read
  - write
  - And anyone that is related to the pointer type.

EXAMPLE I

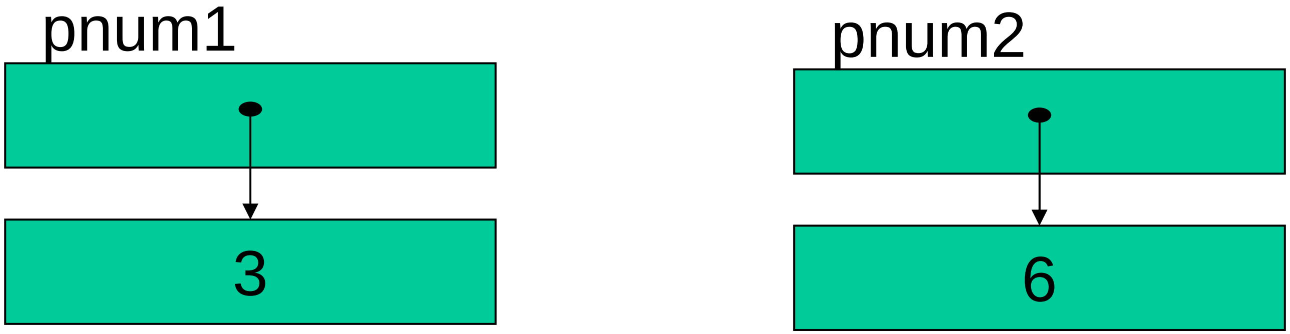
```
...........
TYPE
    tpchar=^char;
VAR
    pchar:tpchar;

BEGIN
.........
New(pchar);
Readln(pchar^); {for instance ‘B’}
pchar^:=Pred(pchar^);
Writeln(pchar^);
..........
END.
```



EXAMPLE II

```
..........
TYPE
    tpnum=^integer;
VAR
    pnum1, pnum2:tpnum;
BEGIN
.........
New(pnum1); New(pnum2);
pnum1^:=2; pnum2^:=4;
pnum2^:=pnum1^+pnum2^;
pnum1^:=pnum2^ DIV 2;
..........
END.
```

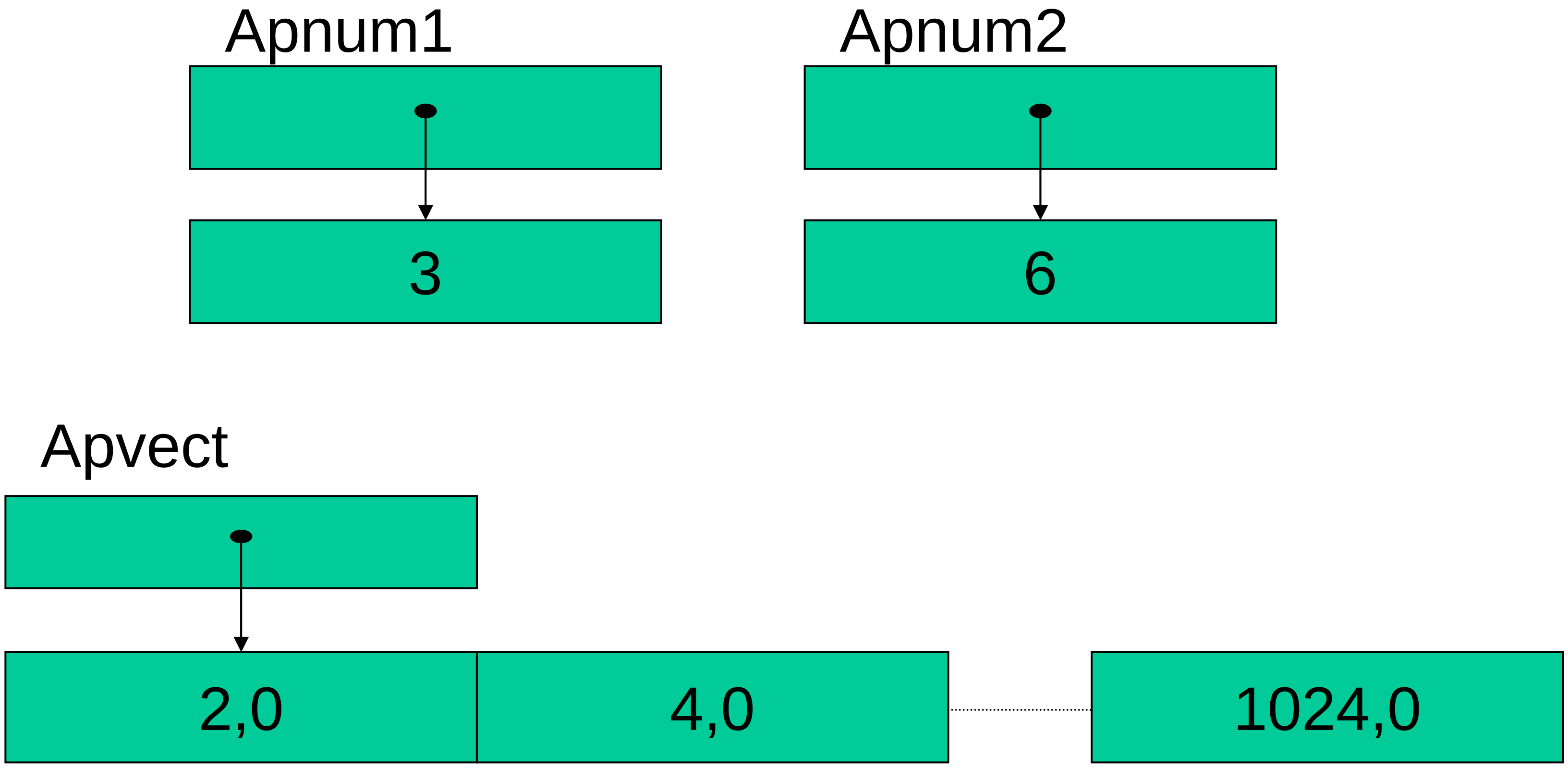


EXAMPLE III

```
...........
TYPE
    tVector10=array[1..10] of real;
    tpnum=^integer;
    tpvector=^tvector10;
VAR
    pnum1, pnum2: tpnum;
    pvect: tpvector10;
    i: integer;
BEGIN
.........
    New(pnum1); New(pnum2); New(pvect);
    pnum1^:=45; pnum2^:=30;
    pvect^[1]=2;
    for i:=2 to 10 do
            pvect^[i]:=pvect^[i-1] * 2;
..........
END.
```

EXAMPLE III

- Tracking the allocations



BASIC POINTERS OPERATIONS

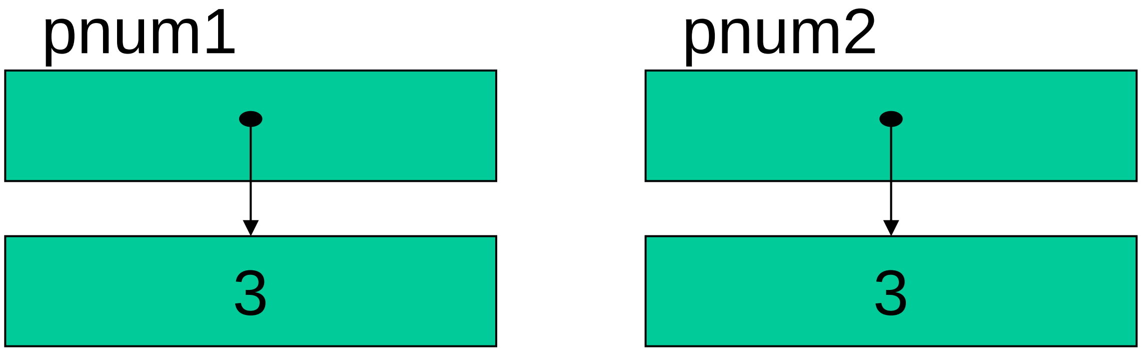
- The allowed operations are:
  - **Comparison:** the addresses contained by the the pointers are compared.

    pnum1=pnum2

  - **Allocation:** the address of the pointer to the left of the expression is allocated to the right one.

    pnum1:=pnum2

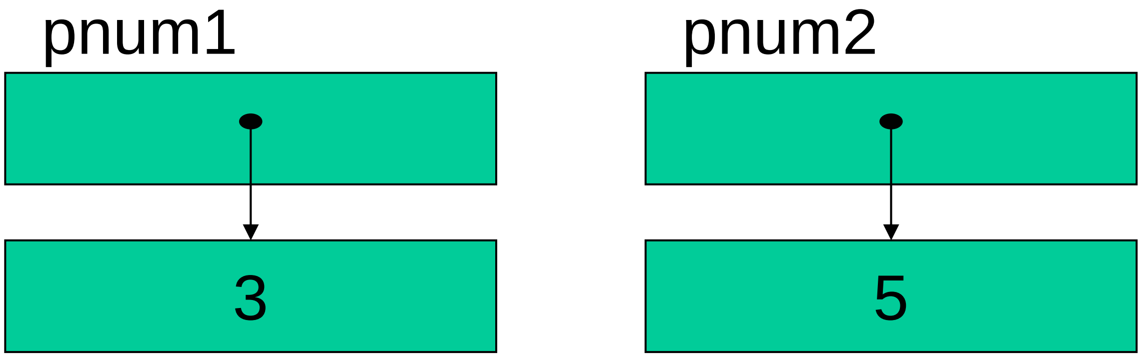
POINTER COMPARISON



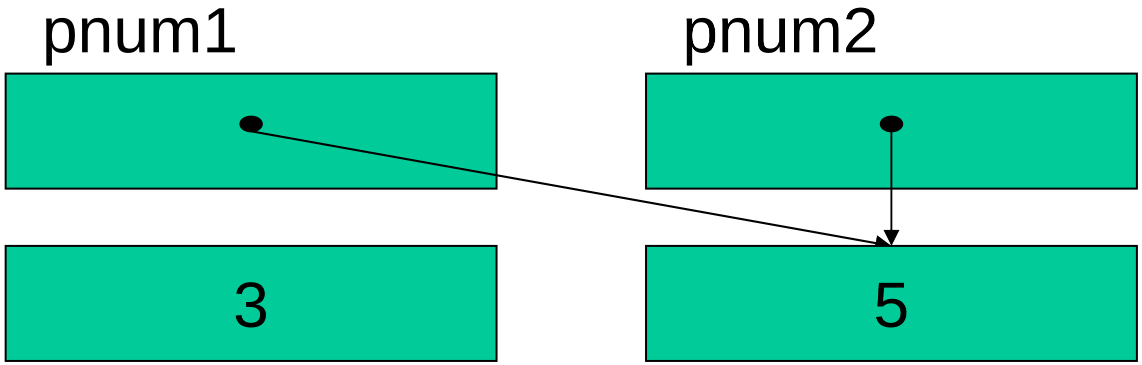
- pnum1=pnum2

- The last comparison between pointers becomes “false” since each one are pointing to different addresses.

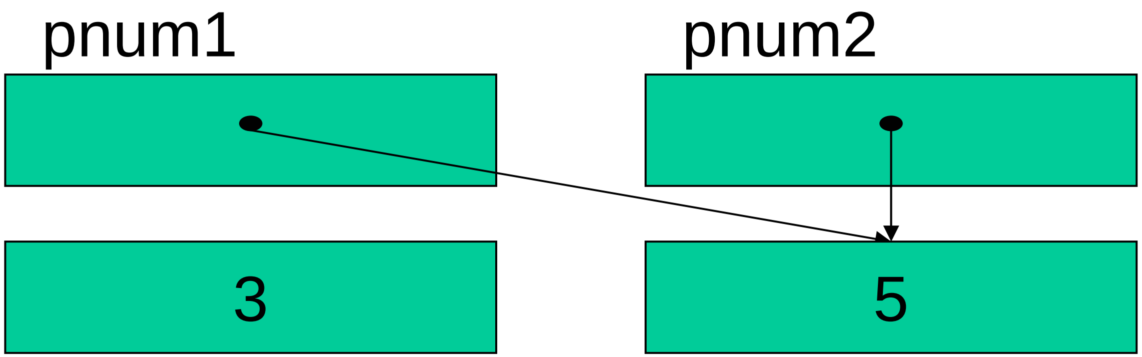
POINTER ALLOCATION



- pnum1:=pnum2



ALLOCATION COLLATERAL EFFECTS



- **Alliasing and Space Memory management:**
  - Any change on pnum1 automatically affect to pnum2.

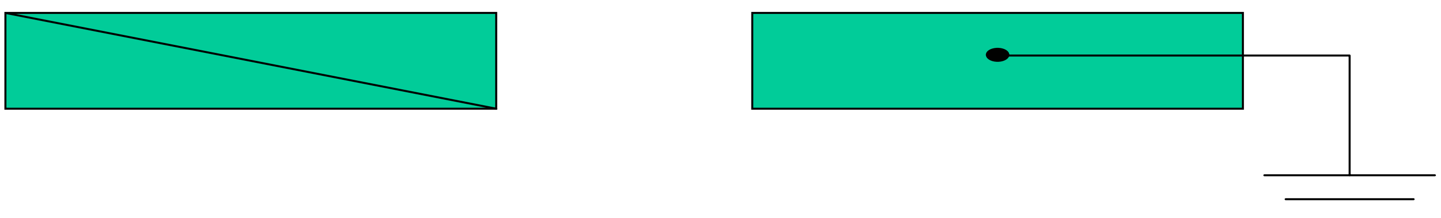
TYPE COHERENCE BETWEEN POINTERS

- Valid operations
  - pnum1:=pnum1
  - pnum1=pnum2
  - pvector1:=pvector2
- Not valid Operations:
  - pnum1:=pchar;
  - pnum1=pvector;

THE NIL VALUE

- A constant in Pascal that can be assigned to a pointer type variable to indicate that the pointer point nothing.

- Graphic representation:



NON RECURSIVE APPLICATIONS USING POINTERS

- One step composed data allocation.
- Composed data as a function output.

ONE STEP COMPOSED DATA ALLOCATION

- To manage the allocation operation when large size data structure are involved.
  - Sorting large size vector.

SORTING LARGE SIZE VECTOR.

```
TYPE
tpFich=^tFich;
tFich=record
    name:string;
    address:string;
    ..............
End; {tFich}
tpstudentlist=array[1..100] of tpFich;
.............
```

- The **sorting** and **searching** operations are made using the pointers.

COMPOSED DATA AS A FUNCTION OUTPUT

- The main idea is to achieve that a function in Pascal returns not only simple data but also composed data structures.

- The solution is to use a pointer instead of the composed data, since a pointer is a simple data.

COMPOSED DATA AS A FUNCTION OUTPUT.

- Program that work out the cartesian coordinates of a point in 2-D, from its polar coordinates.

```
........
TYPE
    tPoint=record
          x,y:real;
    end; {tPunto}
    tpPoint=^tPoint;

VAR
    ang,dist:real;
    orig:tPoint;
    ..........
FUNCTION Cartesian_coordinates(orig:tPoint;ang,dist:real):tpPoint
```

COMPOSED DATA AS A FUNCTION OUTPUT

```
FUNCTION Catersian_coordinates(orig:tPoint;ang,dist:real):tpPoint
VAR
    pPoint:tpPoint;
Begin
    New(pPoint);
    pPoint^.x:=orig.x+dist*cos(ang);
    pPoint^.y:=orig.y+dist*sen(ang):
    Cartesian_coordinates:=pPoint;
End; {Destino}
```