5. Dados dos arreglos a, b : **array**[1..n] **of nat** se dice que a es "lexicográficamente menor" que b sii existe $k \in \{1 \dots n\}$ tal que a[k] < b[k], y para todo $i \in \{1 \dots k-1\}$ se cumple a[i] = b[i]. En otras palabras, si en la primera posición en que a y b difieren, el valor de a es menor que el de b. También se dice que a es "lexicográficamente menor o igual" a b sii a es lexicográficamente menor que b o a es igual a b.

   (a) Escribir un algoritmo lex_less que recibe ambos arreglos y determina si a es lexicográficamente menor que b.

```
fun lex_less(a,b: array[1..n] of nat) ret res: bool
    var i: nat
    i := 1

    while i < n ∧ a[i] = b[i] do
        i := i+1
    od
    res := a[i] < b[i]
end fun
```

   (b) Escribir un algoritmo lex_less_or_equal que recibe ambos arreglos y determina si a es lexicográficamente menor o igual a b.

```
fun lex_less_or_equal(a,b: array[1..n] of nat) ret res: bool
    var i: nat
    i := 1

    while i < n ∧ a[i] = b[i] do
        i := i+1
    od
    res := a[i] ≤ b[i]
end fun
```

   (c) Dado el tipo enumerado

type ord = **enumerate**
igual
menor
mayor
end enumerate

   Escribir un algoritmo lex_compare que recibe ambos arreglos y devuelve valores en el tipo ord. ¿Cuál es el interés de escribir este algoritmo?

```
fun lex_compare(a,b: array[1..n] of nat) ret res: ord
    var i: nat
    i := 1

    while i < n ∧ a[i] = b[i] do
        i := i+1
    od
    if a[i] = b[i] then
        res := igual
    else if a[i] < b[i] then
        res := menor
    else if a[i] > b[i] then
        res := mayor
    fi
end fun
```