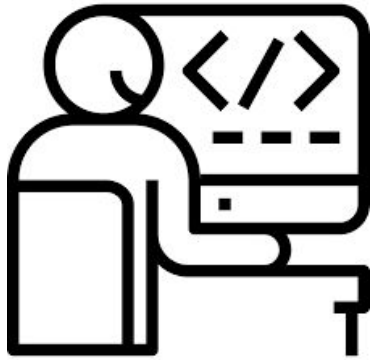


Algoritmos y Programación I

AyPI – **Temas de la clase pasada**

- Tipo de Dato Arreglo

AyPI – **Temas de la clase de hoy**

- Tipos de Datos Arreglo (Continuación)
- Operaciones con vectores

AyPI – TIPOS DE DATOS VECTOR

BÚSQUEDA

La operación de búsqueda implica recorrer el vector buscando un dato que puede o no estar en el vector. El vector puede estar ordenado por el criterio de búsqueda o no.

Vector Sin Orden

Se debe recorrer todo el vector (en el peor de los casos), y detener la búsqueda en el momento que se encuentra el dato buscado o que se terminó el vector.

Vector Con Orden

Se debe aprovechar el orden y detener la búsqueda cuando (suponiendo orden de menor a mayor):

- Se encuentra el dato buscado.
- Se encuentra un valor mayor al buscado
- Se terminó el vector.

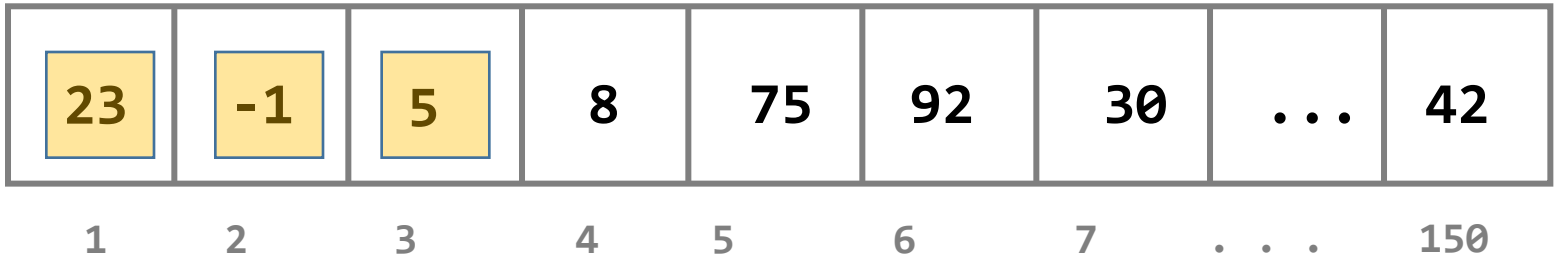
AyPI – TIPOS DE DATOS VECTOR BÚSQUEDA

Vector Sin Orden

`valor = 5`

El valor 5 existe en el vector v.

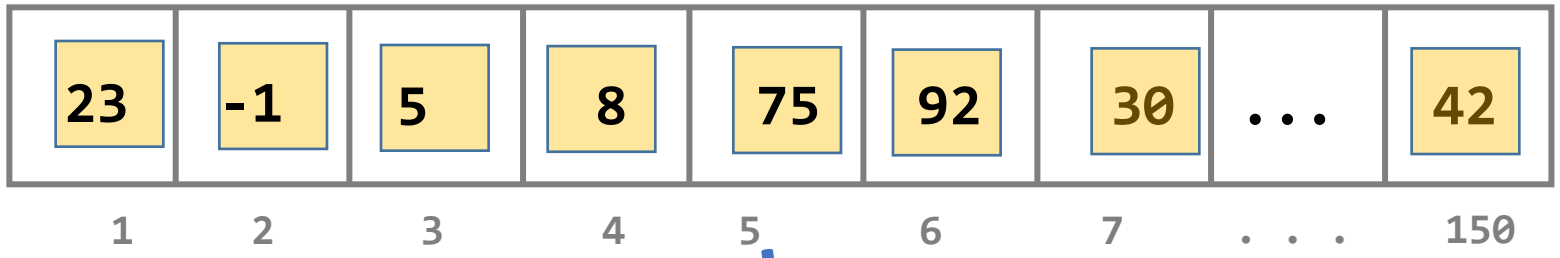
v



`valor = 15`

El valor 15 no existe en el vector v.

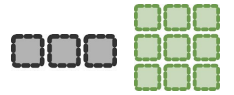
v



¿Qué estructura de control utilizo?

¿Qué tipo de módulo?

AyPI – TIPOS DE DATOS VECTOR BÚSQUEDA

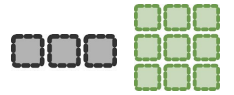


Vector Sin Orden

```
Program uno;
  const
    tam = 150;
  type
    vector = array [1..tam]  of integer;
  var
    v: vector;  num:integer;

  begin
    read (num);
    cargarNumeros (v);
    if (buscarSinOrden (v, num) = true)
    then write (num, ‘ está en el vector’)
    else write (num, ‘ no está en el vector’);
end.
```

AyPI – TIPOS DE DATOS VECTOR BÚSQUEDA



```
function buscarSinOrden (v:vector; valor:integer): boolean;
var
  pos: integer;
  existe: boolean;

Begin
  existe:= false;
  pos:=1;
  while ( (pos <= tam) and (existe <> true) ) do
     if (v[pos]= valor)
       then existe:=true
       else pos:= pos + 1;
  buscarSinOrden:= existe;
end;
```

¿Por qué pos se incrementa en el else?

Ordenación de vectores

Un algoritmo de ordenación es un proceso por el cual un conjunto de elementos es ordenado por algún criterio.

¿Por qué es importante poder ordenar los datos?

Porque las búsquedas tienen un mejor tiempo de respuesta.

Ordenación de vectores

Existen diferentes métodos de ordenación. Cada uno tiene sus ventajas y desventajas en cuanto a facilidad de implementación, tiempo de ejecución, memoria utilizada. Vamos a trabajar con el método de ordenación de ***SELECCIÓN***.

Ordenación de vectores – Método de selección

El ordenamiento por selección busca el valor menor a medida que hace cada pasada y, después de completar la pasada, lo pone en la ubicación correcta.

Después de la primera pasada, el elemento menor quedará ubicado en la primera posición del vector. Después de la segunda pasada, el siguiente elemento menor está en la segunda posición. Este proceso continúa y requiere n−1 pasadas para ordenar los n elementos, ya que el ítem final debe estar en su lugar después de la (n−1)-ésima pasada.

Ordenación de vectores – Método de selección

```
Type vector = array [1..6] of integer;
```

El algoritmo realiza n-1 vueltas, por lo tanto tendrá que realizar 5 vueltas

54	30	15	10	60	20
1	2	3	4	5	6

Primera vuelta

Se debe encontrar el mínimo entre los 6 elementos del vector (posiciones 1 a 6) y colocarlo en la posición 1. El elemento que está en la posición 1 se ubica en la posición del mínimo.

El mínimo es 10 y se encuentra en la posición 4. Entonces hay que intercambiar el contenido de la posición 1 con el contenido de la posición 4.

10	30	15	54	60	20
1	2	3	4	5	6

Ordenación de vectores – Método de selección

```
Type vector = array [1..6] of integer;
```

10	30	15	54	60	20
1	2	3	4	5	6

Segunda vuelta

Se debe encontrar el mínimo entre los 5 elementos del vector (posiciones 2 a 6) y colocarlo en la posición 2. El elemento que está en la posición 2 se ubica en la posición del mínimo.

El mínimo es 15 y se encuentra en la posición 3. Entonces hay que intercambiar el contenido de la posición 2 con el contenido de la posición 3.

10	15	30	54	60	20
1	2	3	4	5	6

Ordenación de vectores – Método de selección

```
Type vector = array [1..6] of integer;
```

10	15	30	54	60	20
1	2	3	4	5	6

Tercera vuelta

Se debe encontrar el mínimo entre los 4 elementos del vector (posiciones 3 a 6) y colocarlo en la posición 3. El elemento que está en la posición 3 se ubica en la posición del mínimo.

El mínimo es 20 y se encuentra en la posición 6. Entonces hay que intercambiar el contenido de la posición 3 con el contenido de la posición 6.

10	15	20	54	60	30
1	2	3	4	5	6

Ordenación de vectores – Método de selección

```
Type vector = array [1..6] of integer;
```

10	15	20	54	60	30
1	2	3	4	5	6

Cuarta vuelta

Se debe encontrar el mínimo entre los 3 elementos del vector (posiciones 4 a 6) y colocarlo en la posición 4. El elemento que está en la posición 4 se ubica en la posición del mínimo.

El mínimo es 30 y se encuentra en la posición 6. Entonces hay que intercambiar el contenido de la posición 4 con el contenido de la posición 6.

10	15	20	30	60	54
1	2	3	4	5	6

Ordenación de vectores – Método de selección

```
Type vector = array [1..6] of integer;
```

10	15	20	30	60	54
1	2	3	4	5	6

Quinta vuelta

Se debe encontrar el mínimo entre los 2 elementos del vector (posiciones 5 a 6) y colocarlo en la posición 5. El elemento que está en la posición 5 se ubica en la posición del mínimo.
El mínimo es 54 y se encuentra en la posición 6. Entonces hay que intercambiar el contenido de la posición 5 con el contenido de la posición 6.

¡El vector tiene sus elementos en orden!

10	15	20	30	54	60
1	2	3	4	5	6

¿Cómo lo implemento?

Ordenación de vectores – Método de selección

```
Type vector = array [1..tam] of integer;

Procedure OrdenarPorSeleccion (var v: vector);
   var i, j, p: integer;
       elemento: integer;
   begin
      for i:=1 to tam-1 do begin
      { busca el mínimo v[p] entre v[i], ..., v[N] }
         p := i;
         for j := i+1 to tam do
             if (v[ j ] < v[ p ]) then p:=j;
         {intercambia v[i] y v[p] }
         elemento := v[ p ];
         v[ p ] := v[ i ];
         v[ i ] := elemento;
      end;
   end;
```

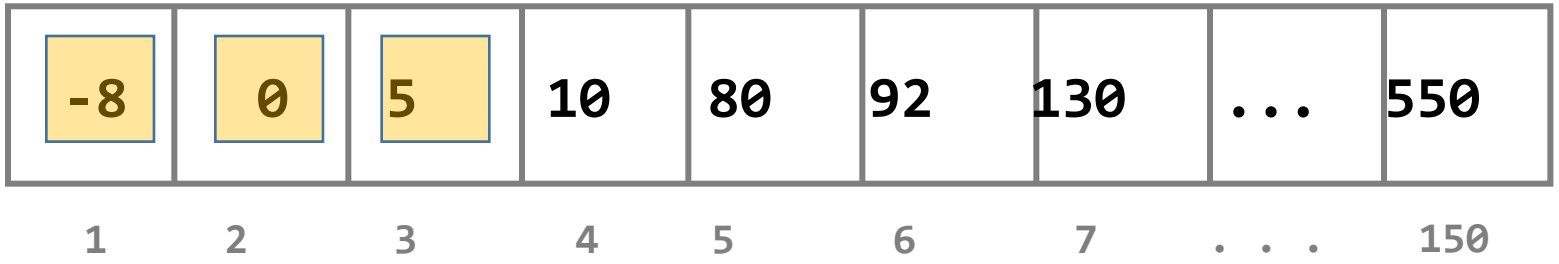
AyPI – TIPOS DE DATOS VECTOR BÚSQUEDA

Vector Con Orden

valor = 5

El valor 5 existe en el vector v.

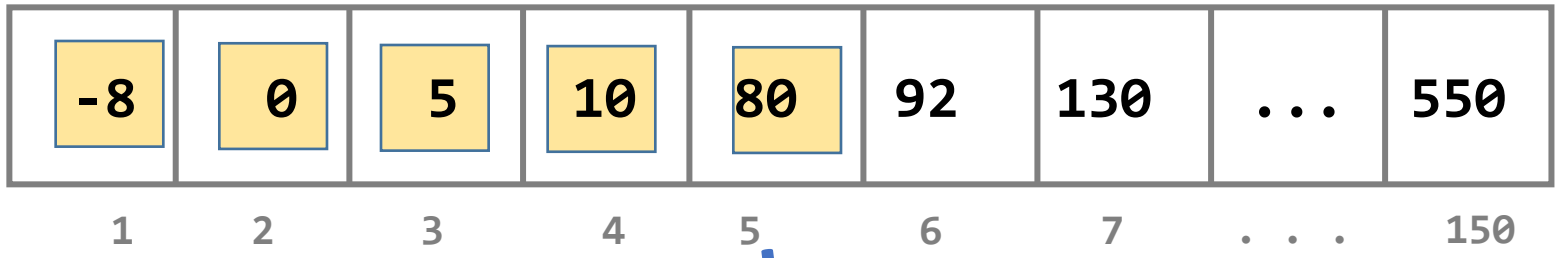
v



valor = 15

El valor 15 no existe en el vector v.

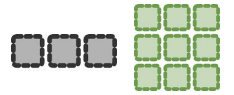
v



¿Qué estructura de control utilizo?

¿Qué tipo de módulo?

AyPI – TIPOS DE DATOS VECTOR BÚSQUEDA



Vector Con Orden

```
Program uno;
  const
    tam = 150;
  type
    vector = array [1..tam]  of integer;
  var
    v: vector;  num:integer;

begin
    read (num);
    cargarNumeros (v);
    if (buscarConOrden(v,num) = true)
    then write (num, ‘ Está en el vector’)
    else write (num, ‘ No está en el vector’);
end.
```

AyPI – TIPOS DE DATOS VECTOR BÚSQUEDA

Vector Ordenado

```
function buscarConOrden (v: vector; valor: integer): boolean;

Var
  pos:integer;

Begin
  pos:=1;
  while ( (pos < tam) and (v[pos]<valor)) do
     pos:= pos + 1;
  if (v[pos]= valor)
  then buscarConOrden:= true
  else buscarConOrden:= false;
end;
```

¿Se puede invertir el orden en la pregunta? NO

¿Debo evaluar las dos condiciones antes de devolver el valor? Sí

AyPI – **TIPOS DE DATOS VECTOR** EJERCITACIÓN

Una inmobiliaria nos encargó un programa para el procesamiento de sus inmuebles. De cada inmueble se deberán considerar las siguientes características: código de identificación del inmueble, tipo, cantidad de habitaciones, cantidad de baños, precio, localidad y fecha de publicación.
Implementar módulos para cada uno de los siguientes ítems:
a. Leer la información de los 200 inmuebles y almacenarlos en un vector.
b. Informar todos los códigos de los inmuebles que tienen más habitaciones que una cantidad que se recibe como parámetro (debe ser leída en el programa principal).
c. Retornar si existe algún inmueble para una localidad que se recibe como parámetro (debe ser leída en el programa principal). Informar el resultado en el programa principal.
d. Retornar el vector ordenado por localidad.
e. A partir del vector obtenido en el ítem d), retornar si existe algún inmueble para una localidad que se recibe como parámetro (debe ser leída en el programa principal). Informar el resultado en el programa principal.
f. A partir del vector obtenido en el ítem d), informar los nombres de las localidades y su cantidad de inmuebles.
g. Retornar las cantidades de inmuebles para cada mes de publicación.
h. Informar lo retornado por el ítem g).

AyPI – TIPOS DE DATOS VECTOR EJERCITACIÓN

```
Program usandoVectores;
const dimF = 200;
type
  fecha = record
            dia: integer;
            mes:integer;
            anio:integer;
          end;
  inmueble = record
               codigo: integer;
               tipo: string;
               cantHab: integer;
               cantBanios:integer;
               precio: real;
               localidad: string;
               fechaPub:fecha;
             end;
  vInmuebles = array[1..dimF] of inmueble;
  vMeses = array[1..12] of integer;


  { Implementacion de los modulos }

  ...
```

```
var v: vInmuebles;
    cantidad: integer;
    m: vMeses;
    localidad: string;
begin
  ModuloA (v);
  read (cantidad);
  ModuloB (v, cantidad);
  read (localidad);
  if (ModuloC (v, localidad) = true)
  then writeln ('Existe al menos un inmueble en la localidad: ', localidad)
  else writeln ('NO existe un inmueble en la localidad: ', localidad);
  ModuloD (v);
  read (localidad);
  if (ModuloE (v, localidad) = true)
  then writeln ('Existe al menos un inmueble en la localidad: ', localidad)
  else writeln ('NO existe un inmueble en la localidad: ', localidad);
  ModuloF (v);
  ModuloG (v, m);
  ModuloH (m);
end.
```

a. Leer la información de los 200 inmuebles y almacenarlos en un vector.

```
procedure LeerFecha (var f: fecha);
begin
  read (f.dia);
  read (f.mes);
  read (f.anio);
end;

procedure LeerInmueble (var inmu: inmueble);
begin
  read (inmu.codigo);
  read (inmu.tipo);
  read (inmu.cantHab);
  read (inmu.cantBanios);
  read (inmu.precio);
  read (inmu.localidad);
  LeerFecha (inmu.fechaPub);
end;

procedure ModuloA (var v: vInmuebles);
var inmu: inmueble;
    i: integer;
begin
  for i:= 1 to dimF do
  begin
    LeerInmueble (inmu);
    v[i]:= inmu;
  end;
end;
```

b. Informar todos los códigos de los inmuebles que tienen más habitaciones que una cantidad que se recibe como parámetro (debe ser leída en el programa principal).

```
procedure ModuloB (var v: vInmuebles; unaCantidad: integer);
var i: integer;
begin
  for i:= 1 to dimF do
  begin
    if (v[i].cantHab > unaCantidad)
    then writeln (v[i].codigo);
  end;
end;
```

c. Retornar si existe algún inmueble para una localidad que se recibe como parámetro (debe ser leída en el programa principal). Informar el resultado en el programa principal.

```
function ModuloC (v: vInmuebles; unaLocalidad: string): boolean;
var pos:integer;
    existe: boolean;
Begin
  existe:= false;
  pos:=1;
  while ( (pos <= dimF) and (existe <> true) ) do
     if (v[pos].localidad = unaLocalidad)
       then existe:=true
       else pos:= pos + 1;
  ModuloC:= existe;
end;
```

d. Retornar el vector ordenado por localidad.

```
procedure ModuloD (var v: vInmuebles);
var i, j, p: integer;
    elemento: inmueble;
begin
  for i:= 1 to dimF - 1 do
  begin
    { busca el mínimo v[p] entre v[i], ..., v[N] }
      p := i;
      for j := i+1 to dimF do
      if (v[ j ].localidad < v[ p ].localidad) then p:=j;
    {intercambia v[i] y v[p] }
      elemento := v[ p ];
      v[ p ] := v[ i ];
      v[ i ] := elemento;
  end;
end;
```

e. A partir del vector obtenido en el ítem d), retornar si existe algún inmueble para una localidad que se recibe como parámetro (debe ser leída en el programa principal). Informar el resultado en el programa principal.

```
function ModuloE (v: vInmuebles; unaLocalidad: string): boolean;
var pos:integer;
begin
  pos:=1;
  while ( (pos < dimF) and (v[pos].localidad < unaLocalidad)) do
     pos:= pos + 1;
  if (v[pos].localidad = unaLocalidad)
  then ModuloE:= true
  else ModuloE:= false;
end;
```

f. A partir del vector obtenido en el ítem d), informar los nombres de las localidades y su cantidad de inmuebles.

```
procedure ModuloF (v: vInmuebles);
var i, cant: integer;
    locActual: string;
begin
  i:= 1;
  while (i <= dimF) do
  begin
    locActual:= v [i].localidad;
    cant:= 0;
    while ((i <= dimF) and (locActual = v [i].localidad)) do
    begin
      cant:= cant + 1;
      i:= i + 1;
    end;
    writeln ('Localidad: ', locActual, ' Cantidad: ', cant);
  end;
end;
```

g. Retornar las cantidades de inmuebles para cada mes de publicación.

```
procedure InicializarVector (var m: vMeses);
var i: integer;
begin
  for i:= 1 to 12 do
     m[i]:= 0;
end;

procedure ModuloG (v: vInmuebles; var m: vMeses);
var inmu: inmueble;
    i: integer;
begin
  InicializarVector (m);
  for i:= 1 to dimF do
  begin
    inmu:= v[i];
    m [ inmu.fechaPub.mes ] :=  m [ inmu.fechaPub.mes ] + 1;
  end;
end;
```

h. Informar lo retornado por el ítem g).

```
procedure ModuloH (m: vMeses);
var i: integer;
begin
  for i:= 1 to 12 do
    writeln ('La cantidad de inmuebles publicados en el mes ', i, ' es ', m [i])
end;
```