# TEMAS DE INTRO I

- Pilas (datos/estructuras de control)
- **Filas (datos/estructuras de control)**
- Modularización y Parámetros
- Variables
- Método de Desarrollo y  Funciones
- Arreglos
- Matrices

**PILAS**

| | |
|---|---|
| Receta | Programa |
| Ingredientes | Datos |
| Pasos | Control, Secuencias |
| Chef | Procesador |
| Idioma | Lenguaje de programación |
| Una elaboración | Proceso |

**SECUENCIAS;**
**SELECCIÓN;**
**ITERACIÓN**

**PASCAL**

```
Program DivideMenores10;
{este programa ……..}

{$INCLUDE /IntroProg/Estructu}
var
  Origen, MenorIgual, Mayores: Pila;
 Begin
    ReadPila(Origen);
    InicPila(MenorIgual, ' ');
    InicPila(Mayores, ' ');
    While not PilaVacia(Origen) do
         if tope(Origen) <= 10
           then
             Apilar (MenorIgual, Desapilar(Origen))
          else
             Apilar (Mayores, Desapilar(Origen));
    writePila(MenorIgual);
    writePila(Mayores);
end.
```

```
Program DivideMenores10;
{este programa pasa todos los números de la pila  Origen a la pila
MenorIgual si son menores o iguales a 10  y sino los pasa a
 Mayores}

{$INCLUDE /IntroProg/Estructu}
var
  Origen, MenorIgual, Mayores: Pila;
 Begin
    ReadPila(Origen);
    InicPila(MenorIgual, ' ');
    InicPila(Mayores, ' ');
    While not PilaVacia(Origen) do
         if tope(Origen) <= 10
           then
             Apilar (MenorIgual, Desapilar(Origen))
          else
             Apilar (Mayores, Desapilar(Origen));
    writePila(MenorIgual);
    writePila(Mayores);
end.
```

```
Program DivideMenores10;

{este programa pasa todos los números de la pila  Origen a la pila
MenorIgual si son menores o iguales a 10, sino los deja en
Origen}
```

```
Program DivideMenores10;
{este programa pasa todos los números de la pila  Origen a la pila
MenorIgual si son menores o iguales a 10, sino los deja en
Origen}

{$INCLUDE /IntroProg/Estructu}
var
  Origen, MenorIgual:Pila;
 Begin
    ReadPila(Origen);
    InicPila(MenorIgual, ' ');
    While not PilaVacia(Origen) do
         if tope(Origen) <= 10
           then
             Apilar (MenorIgual, Desapilar(Origen));
   writePila(MenorIgual);
end.
```

¿ES CORRECTO?

```
Program DivideMenores10;
{este programa pasa todos los números de la pila  Origen
a la pila MenorIgual si son menores o iguales a 10, sinp queda en O

{$INCLUDE /IntroProg/Estructu}
var
  Origen, MenorIgual, Aux:Pila;
 Begin
    ReadPila(Origen);
    InicPila(MenorIgual, ' ');
    While not PilaVacia(Origen) do
         if tope(Origen) <= 10
         then
             Apilar (MenorIgual, Desapilar(Origen))
        else
             Apilar (Aux, Desapilar(Origen));
    While not PilaVacia(Aux) do
             Apilar (Origen, Desapilar(Aux));
    writePila(MenorIgual);
     writePila(Origen);
end.
```

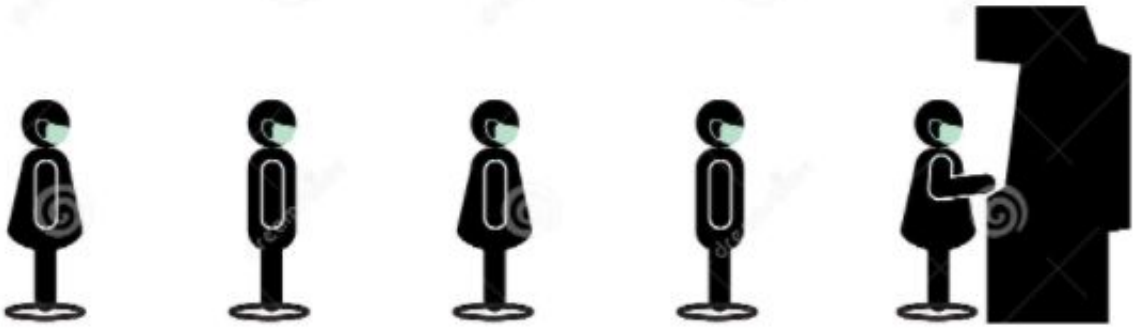
**FILAS**

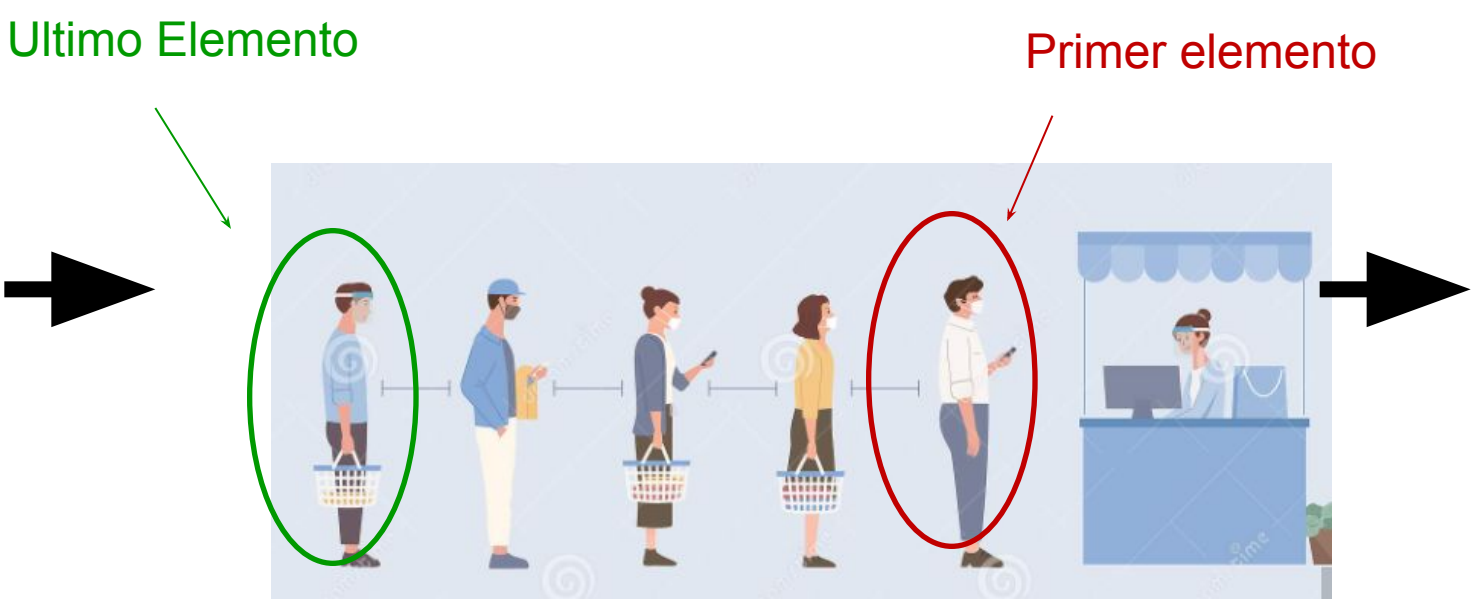
| | |
|---|---|
| Receta | Programa |
| Ingredientes | Datos |
| Pasos | Control, Secuencias |
| Chef | Procesador |
| Idioma | Lenguaje de programación |
| Una elaboración | Proceso |

**CONDICIONES COMPUESTAS en SELECCIÓN; ITERACIÓN**

**PASCAL**

# Filas






Una **Fila o Cola** es una colección ordenada de elementos homogéneos en el que los nuevos elementos se añaden de a uno por un extremo (el final) y se quita de a uno por el otro extremo (el frente).

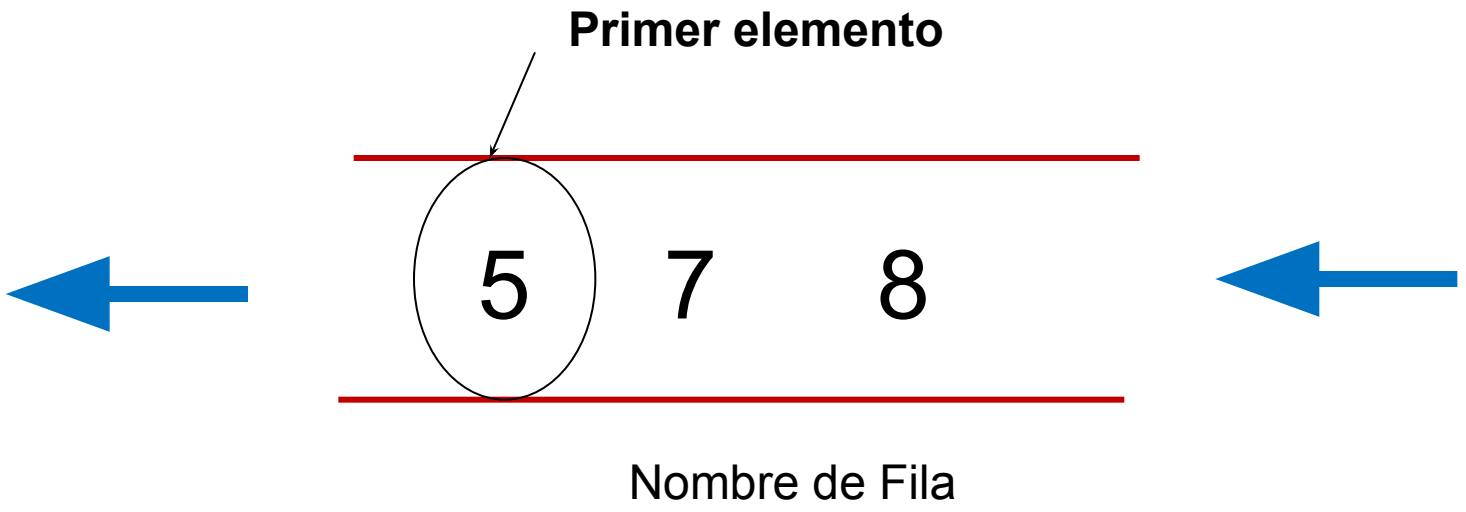
En las **Filas** el elemento que **entró primero también sale primero**.

# Filas en Pascal

- Fila de enteros

- Se identifican con un  nombre

- Sólo se puede  acceder al primer elemento (que es el primero que entró)



# Operaciones sobre Filas

| objetivo | Fila |
|---|---|
| Definición | **NombreFila: Fila** |
| Inicialización estática | **InicFila(NombreFila, ´1 3 4´)** |
| Carga por teclado | **ReadFila(NombreFila)** |
| Vacia? → ( SI / NO) | **FilaVacia(NombreFila)** |
| PrimerElemento → <nro> | **Primero(NombreFila)** |
| RecuperarElemento → <nro> | **Extraer(NombreFila)** |
| AgregarElemento | **Agregar(NombreFila, '<nro>' )** |
| MostrarElementos | **WriteFila(NombreFila)** |

```
C:\fpc\bin>ejemploCompara
Ingresar elementos a la fila:    <Primero> <  ...  > <Ultimo>
1 3 4
```

# Operaciones sobre Filas

| objetivo | Fila |
|---|---|
| Definición | **NombreFila: Fila** |
| Inicialización estática | **InicFila(NombreFila, ´1 3 4)** |
| Carga por teclado | **ReadFila(NombreFila)** |
| Vacia? → ( SI / NO) | **FilaVacia(NombreFila)** |
| PrimerElemento → <nro> | **Primero(NombreFila)** |
| RecuperarElemento → <nro> | **Extraer(NombreFila)** |
| AgregarElemento | **Agregar(NombreFila, '<nro>' )** |
| MostrarElementos | **WriteFila(NombreFila)** |

**Agregar(NombreFila, Extraer(NombreOtraFila ) )**

# Problema:

Se quieren pasar los elementos de una Fila llamada Origen a otra Fila llamada Destino.

```
Program PasaElementosFila;
{Este Programa pasa los elementos de la fila Origen a Destino}

{$INCLUDE /IntroProg/Estructu}

var
  Origen, Destino: Fila;

Begin
  ReadFila(Origen);
  InicFila(Destino, ' ');
  While not FilaVacia(Origen) do
      Agregar(Destino, Extraer(Origen));
  WriteFila(Origen);
  WriteFila(Destino)
end.
```

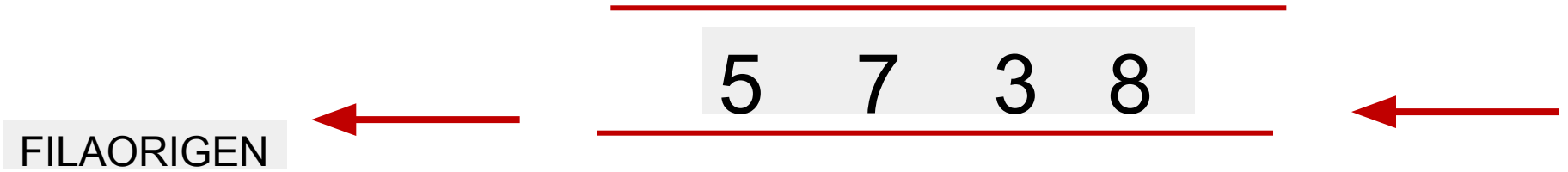
```
Program PasaElementosFila;
{Este Programa pasa los elementos de la fila Origen a Destino}
{$INCLUDE /IntroProg/Estructu}
var
  Origen, Destino: Fila;
 Begin
  ReadFila(Origen);
  InicFila(Destino, ' ');
  While not FilaVacia(Origen) do
       Agregar(Destino, Extraer(Origen));
  WriteFila(Origen);
  WriteFila(Destino)
end.
```

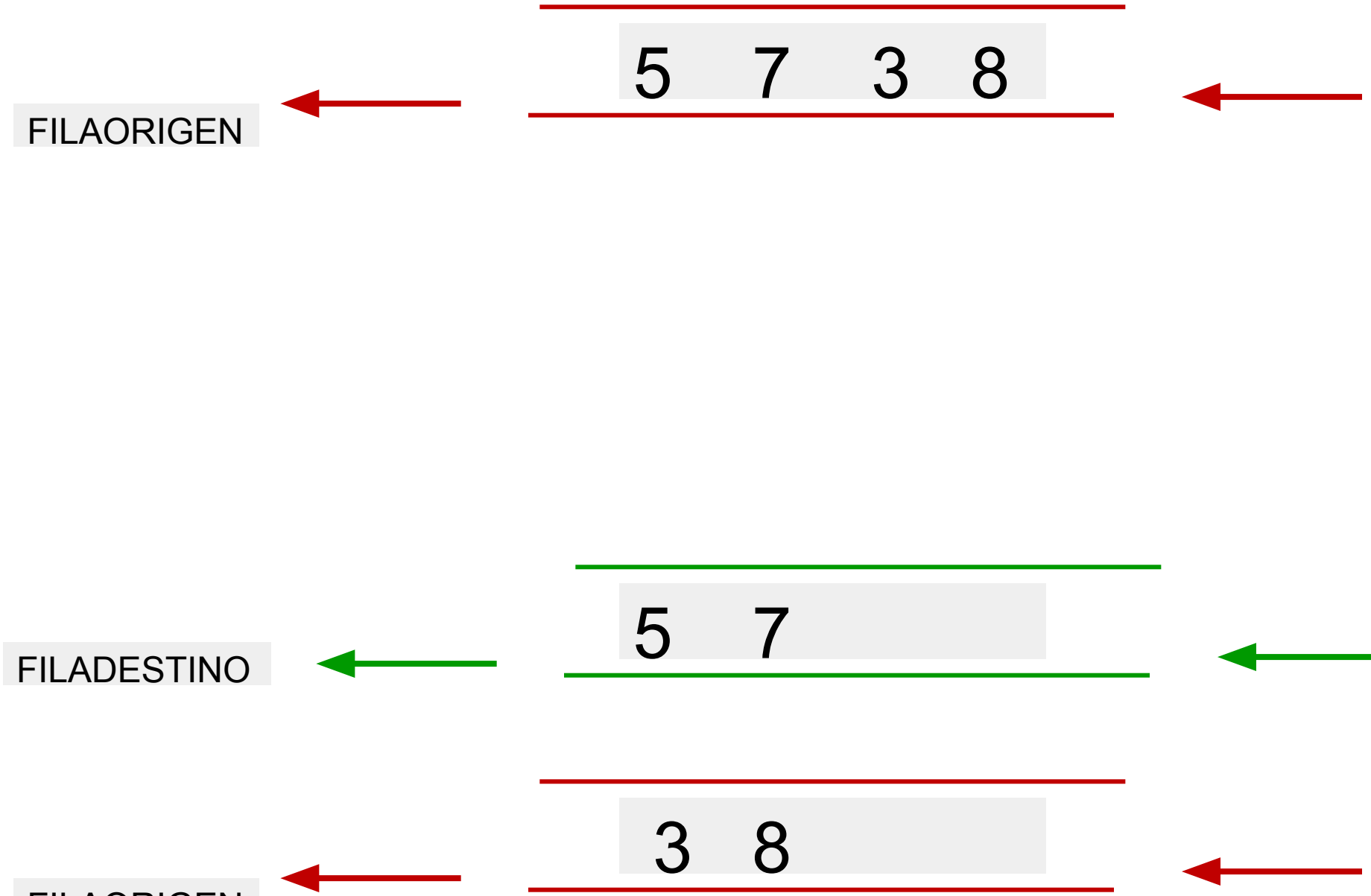
**Origen <primero>  3 6 19 <último>**

**Destino <primero>  3 6 19 <último>**

# Problema 2

Pasar los elementos de una Fila Origen a la fila Destino hasta encontrar un número 3


5 7 3 8
FILAORIGEN


5 7 3 8
FILAORIGEN
5 7
FILADESTINO
3 8
FILAORIGEN

```
Program  PasajeParcial;
  { Este programa Pasa los elementos de la Fila origen a la fila
    destino hasta llegar al numero 3 }
{$INCLUDE /IntroProg/Estructu}
Var
FilaOrigen, FilaDestino: Fila;
Begin
   ReadFila(FilaOrigen);
   inicFila(FilaDestino, ' ' );
   while (primero(FilaOrigen) <> 3)do
        Agregar(FilaDestino, extraer(FilaOrigen));
   WriteFila(FilaOrigen);
   WriteFila(FilaDestino);
end.
```

¿Es correcta la solución?

```
Program  PasajeParcial;
  { Este programa Pasa los elementos de la Fila origen a la fila
    destino hasta llegar al numero 3 }
{$INCLUDE /IntroProg/Estructu}
Var
FilaOrigen, FilaDestino: Fila;
Begin
   ReadFila(FilaOrigen);
   inicFila(FilaDestino, ' ' );
   while (primero(FilaOrigen) <> 3)do
        Agregar(FilaDestino, extraer(FilaOrigen));
   WriteFila(FilaOrigen);
   WriteFila(FilaDestino);
end.
```

¿Es correcta la solución?

Si la FilaOrigen no tiene el 3 DA ERROR !!!

```
Program  PasajeParcial;
  { Este programa Pasa los elementos de la Fila origen a la fila destino
    hasta llegar al numero 3 }

{$INCLUDE /IntroProg/Estructu}
Var
FilaOrigen, FilaDestino: Fila;
Begin
   ReadFila(FilaOrigen);
   inicFila(FilaDestino, ' ' );
   while not Filavacia(FilaOrigen) do
       If  (primero(FilaOrigen) <> 3) then
              Agregar(FilaDestino, extraer(FilaOrigen));
   WriteFila(FilaOrigen);
   WriteFila(FilaDestino);
end.
```

¿Es correcta la solución?

Si hay un elemento 3 no sale del ciclo ! ! !

# Condiciones múltiples

Hay situaciones en que se desean evaluar varias condiciones a la vez:

Una condición  múltiple es una secuencia de condiciones simples que se unen con conectores lógicos (AND  ///  OR)

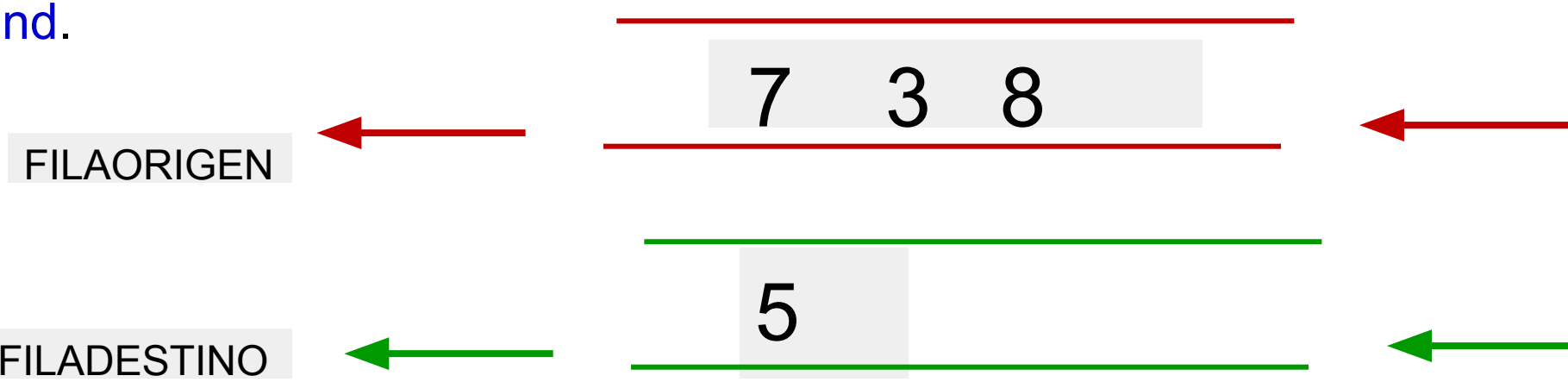
```
Program  PasajeParcial;
  { Este programa Pasa los elementos de la Fila origen a
    la fila destino hasta llegar al número 3 }
..........
```

Tenemos que pasar los elementos de Origen mientras no esté vacía y no encontremos el número 3.

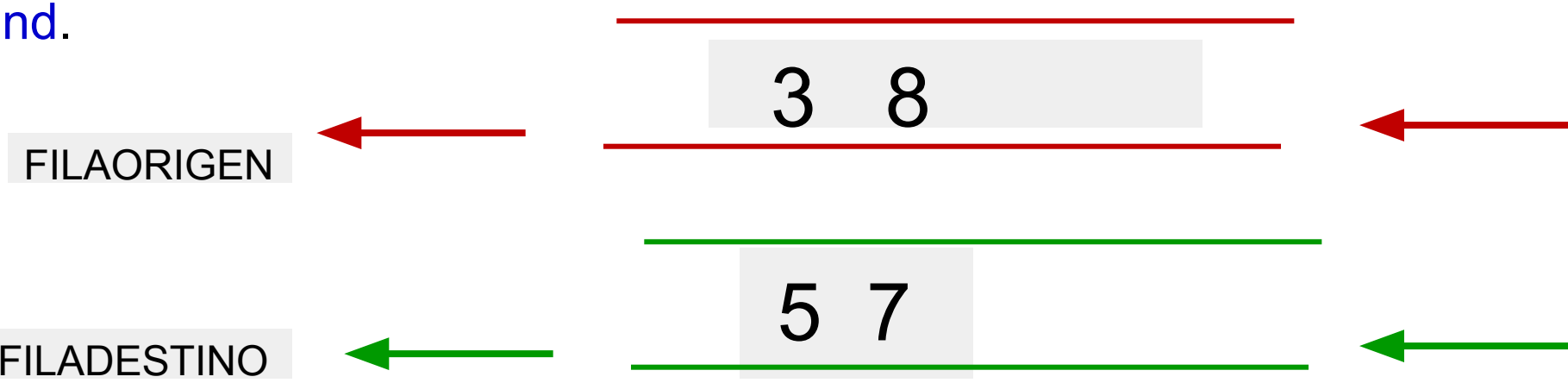
```
while not Filavacia(FilaOrigen) and (primero(FilaOrigen))<> 3 do
```

```
Program  PasajeParcial;
 {Pasa los elementos de la Fila origen a la fila destino hasta llegar al 3 }
{$INCLUDE /IntroProg/Estructu}

Var
FilaOrigen, FilaDestino: Fila;

Begin
   ReadFila(FilaOrigen);
   inicFila(FilaDestino, ' ' );
   while not Filavacia(FilaOrigen) and (primero(FilaOrigen) <> 3)do
        Agregar(FilaDestino, extraer(FilaOrigen));
   WriteFila(FilaOrigen);
   WriteFila(FilaDestino);
end.
```

```
Program  PasajeParcial;
 {Pasa los elementos de la Fila origen a la fila destino hasta llegar al 3 }
{$INCLUDE /IntroProg/Estructu}

Var
FilaOrigen, FilaDestino: Fila;

Begin
   ReadFila(FilaOrigen);
   inicFila(FilaDestino, ' ' );
   while not Filavacia(FilaOrigen) and (primero(FilaOrigen) <> 3)do
       Agregar(FilaDestino, extraer(FilaOrigen));
   WriteFila(FilaOrigen);
   WriteFila(FilaDestino);
end.
```



```
Program  PasajeParcial;
 {Pasa los elementos de la Fila origen a la fila destino hasta llegar al 3 }
{$INCLUDE /IntroProg/Estructu}

Var
FilaOrigen, FilaDestino: Fila;

Begin
   ReadFila(FilaOrigen);
   inicFila(FilaDestino, ' ' );
   while not Filavacia(FilaOrigen) and (primero(FilaOrigen) <> 3)do
       Agregar(FilaDestino, extraer(FilaOrigen));
   WriteFila(FilaOrigen);
   WriteFila(FilaDestino);
end.
```



**Tablas de Verdad**

## Not (No)

| Condición 1 | Not Condición 1 |
|---|---|
| Falso | Verdadero |
| Verdadero | Falso |

not Filavacia(FilaOrigen)

**Tablas de Verdad** And (y)

| Condición 1 | Condición 2 | Condición 1 and Condición 2 |
|---|---|---|
| Verdadero | Verdadero | Verdadero |
| Verdadero | Falso | Falso |
| Falso | Verdadero | Falso |
| Falso | Falso | Falso |

```
while not Filavacia(FilaOrigen) and (primero(FilaOrigen) <> 3)do
```

**Tablas de Verdad** Or (o)

| Condición 1 | Condición 2 | Condición 1 or Condición 2 |
|---|---|---|
| Verdadero | Verdadero | Verdadero |
| Verdadero | Falso | Verdadero |
| Falso | Verdadero | Verdadero |
| Falso | Falso | Falso |

## Tablas de Verdad: OR

| Condición 1 | Condición 2 | Condición 1 or Condición 2 |
|---|---|---|
| Verdadero | Verdadero | Verdadero |
| Verdadero | Falso | Verdadero |
| Falso | Verdadero | Verdadero |
| Falso | Falso | Falso |

Volviendo al problema de pasar los elementos hasta encontar un 3, esta condición es correcta?

```
while not Filavacia(FilaOrigen) or (primero(FilaOrigen) <> 3)do
```

## Cuando finaliza este while?

while not Filavacia(Origen) and  not Filavacia(Destino) do

Cuando Origen sea vacía Ó Destino sea vacía Ó ambas sean vacías

## Cuando finaliza este while?

while not Filavacia(Origen) OR not Filavacia(Destino) do

Cuando ambas sean vacías

```
Program  PasajeParcial;
  { Este programa Pasa los elementos de la Fila origen a la fila
    destino hasta llegar al numero 3 }
{$INCLUDE /IntroProg/Estructu}
Var
FilaOrigen, FilaDestino: Fila;
Begin
   ReadFila(FilaOrigen);
   inicFila(FilaDestino, ' ' );
   while (primero(FilaOrigen) <> 3) and (not Filavacia(FilaOrigen))do
       Agregar(FilaDestino, extraer(FilaOrigen));
   WriteFila(FilaOrigen);
   WriteFila(FilaDestino);
end.
```

¡MAL! **IMPORTA EL ORDEN, no se puede preguntar por el primer elemento de una Fila vacía**

```
Program  PasajeParcial;
  { Este programa Pasa los elementos de la Fila origen a la fila
    destino hasta llegar al numero 3 }
{$INCLUDE /IntroProg/Estructu}
Var
FilaOrigen, FilaDestino: Fila;
Begin
   ReadFila(FilaOrigen);
   inicFila(FilaDestino, ' ' );
   while (not Filavacia(FilaOrigen)) and (primero(FilaOrigen) <> 3) do
       Agregar(FilaDestino, extraer(FilaOrigen));
   WriteFila(FilaOrigen);
   WriteFila(FilaDestino);
end.
```

**Tablas de Verdad**

# Leyes de DeMorgan

| Condición _1 | Condición_ 1 | Not (Condición _1 OR Condición_2) | not condición_1 and not Condición_2 |
|---|---|---|---|
| Verdadero | Verdadero | Falso | Falso |
| Verdadero | Falso | Falso | Falso |
| Falso | Verdadero | Falso | Falso |
| Falso | Falso | Verdadero | Verdadero |

**Tablas de Verdad**

# Leyes de DeMorgan

| Condición _1 | Condición_ 2 | Not (Condición_1 AND Condición_2) | not Condición _1 or not Condición_2 |
|---|---|---|---|
| Verdadero | Verdadero | Falso | Falso |
| Verdadero | Falso | Verdadero | Verdadero |
| Falso | Verdadero | Verdadero | Verdadero |
| Falso | Falso | Verdadero | Verdadero |

# Problema

- Analizar si dos filas tienen la misma cantidad de elementos. Si es así, poner el tope de la pila RESUL en la pila IGUAL sino poner el tope de RESUL en la Pila DISTINTO.

  - **Entender el problema y su objetivo**
  - Entender que datos se manejan
  - Pensar en una estrategia para resolverlo
  - Y LUEGO Codificar

# Problema

- Analizar si dos filas tienen la misma cantidad de elementos. Si es así, poner el tope de la pila RESUL en la pila IGUAL sino poner el tope de RESUL en la Pila DISTINTO.
    - Entender el problema y su objetivo
    - **Entender que datos se manejan**
    - Pensar en una estrategia para resolverlo
    - Y LUEGO Codificar

2 Filas (Origen1 y Origen2) y 3 Pilas (Resul, Igual, Distinto)...
y puede haber más auxiliares

# Problema

- Analizar si dos filas tienen la misma cantidad de elementos. Si es así, poner el tope de la pila RESUL en la pila IGUAL sino poner el tope de RESUL en la Pila DISTINTO.

  - Entender el problema y su objetivo
  - Entender que datos se manejan
  - **Pensar en una estrategia para resolverlo**
  - Y LUEGO Codificar

```
Program  ComparaCantidadElementos;
 {Este programa analiza si dos filas tienen la misma cantidad de elementos. Si es así, el programa
pone  el tope de la pila RESUL en la pila IGUAL sino pone el tope de AUX en la Pila DISTINTO. }
{$INCLUDE /IntroProg/Estructu}
Var
 Origen1, Origen2, Descarte: Fila;
 Resul, Igual, Distinta: Pila;
Begin
   ReadFila(Origen1);
   ReadFila(Origen2);
   InicFila(Descarte, ' ' );
   InicPila(Resul, '1' );
   InicPila(Igual, ' ' );
   InicPila(Distinta, ' ' );
    while not (Filavacia(Origen1)) and not (Filavacia(Origen2)) do
    begin
        Agregar(Descarte, extraer(Origen1));
        Agregar(Descarte, extraer(Origen2));
    end;
   if  (Filavacia(Origen1)) and (Filavacia(Origen2)) then
      Apilar(Igual, Desapilar(Resul))
  else
      Apilar(Distinta, Desapilar(Resul));
 WritePila(Igual);
 WritePila(Distinta);
end.
```

```
Program  ComparaCantidadElementos;
 {Este programa analiza si dos filas tienen la misma cantidad de elementos. Si es así, el programa
pone  el tope de la pila RESUL en la pila IGUAL sino pone el tope de AUX en la Pila DISTINTO. }
{$INCLUDE /IntroProg/Estructu}
Var
 Origen1, Origen2, Descarte: Fila;
 Resul, Igual, Distinta: Pila;
Begin
   .......
   while not (Filavacia(Origen1)) and not (Filavacia(Origen2)) do
    begin
        Agregar(Descarte, extraer(Origen1));
        Agregar(Descarte, extraer(Origen2));
    end;
   if  (Filavacia(Origen1)) and (Filavacia(Origen2)) then
       Apilar(Igual, Desapilar(Resul))
   else
       Apilar(Distinta, Desapilar(Resul));
 WritePila(Igual);
 WritePila(Distinta);
end.
```