☞ Si **P S Q**,

- la descripción de acciones, algoritmo S, y
- las descripciones de estados P y Q

están <u>fuertemente</u> relacionadas

S se define como la descripción de la acción que elimina la diferencia entre P y Q

1

Enlace entre los puntos 1 y 2 del tema

¿qué hace el siguiente algoritmo **S**:

algoritmo S (**DATOS** x, y: entero;
RESULTADOS x, y: entero)

¿Cómo lo hace?

P= {x=X ∧ y=Y}

descripciones de <u>estados</u>

descripciones de <u>acciones</u>

Q= { x=X ∧ y=Y }

instrucción vacía

¿ qué hace **S** ?

Nada

2

2. Notación para el diseño de algoritmos

LENGUAJE ALGORÍTMICO

Instrucción S :

descripción (enunciado) de la acción que elimina la diferencia entre **P** y **Q**

En el ejemplo anterior,
la descripción de la acción **"no hacer nada"**

- se nota con **vacía** y
- se denomina **instrucción vacía**

3

2. Notación para el diseño de algoritmos

```
algoritmo S   (DATOS dato1:tipo_dato1;...;datoN:tipo_datoN;
               RESULTADOS res1:tipo_res1;...;resN:tipo_resN)
tipo
        auxiliar_t1=....; ... ; auxiliar_tn=....;
ftipo
var
        auxiliar_1: auxiliar_t1; ... ; auxiliar_n: auxiliar_tn;
fvar
  P={...}
instrucción_1;
        ...
instrucción _n;
  Q={...}
falgoritmo
```

4

2. Notación para el diseño de algoritmos

- **Sintaxis de instrucciones**
  - **instrucción vacía** `vacía`
  - **instrucción asignación** `:=`
  - **instrucción condicional**

```
opción
        caso1: I1;
        ....
        casoN :IN;
fopción
```

  - **instrucción composición secuencial** `I1;I2;`
  - **instrucción iterativa**

```
mientras B hacer
        I1;
        ....
        IN;
fmientras
```

2. Notación para el diseño de algoritmos

- **Semántica de instrucciones**

P instrucción Q

P vacía; P=Q

Aquella cuya ejecución **NO** provoca ninguna transformación de estados

6

2.1. Asignación :=

- **Sintaxis de la asignación**

  y:=e;

donde

- y es una variable
- e es una expresión evaluable de tipo compatible con y

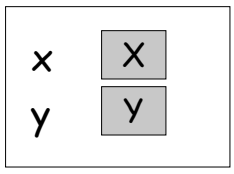
7

2.1. Asignación

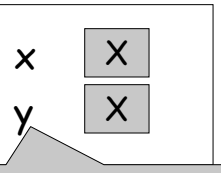
Ejemplo de <u>semántica</u> de la asignación

algoritmo copia (**DATOS** x, y : entero;
RESULTADOS y: entero)

P={x=X ∧ y=Y**}** copia **Q={**x=X ∧ y=X**}**



y:=x;



y **hereda** el valor de x

2.1. Asignación

- **Semántica de la asignación**

  P={e cumple la propiedad **R}**

  y:=e;

  Q={ y cumple la propiedad **R}**

2.1. Asignación

- **Comprobación de la corrección de** copia

algoritmo copia (**DATOS** x, y : entero;
RESULTADOS y: entero)

P={x=X ∧ y=Y**}**

La variable **x** cumple **R** ={x tiene el valor X}

1

y:=x;

2

Q={x=X ∧ y=X**}**

La variable **y** cumple **R** ={y tiene el valor X}

2.2. Composición secuencial

- **Sintaxis de la composición secuencial**

  I1;I2;

donde

- I1 **e** I2 son instrucciones

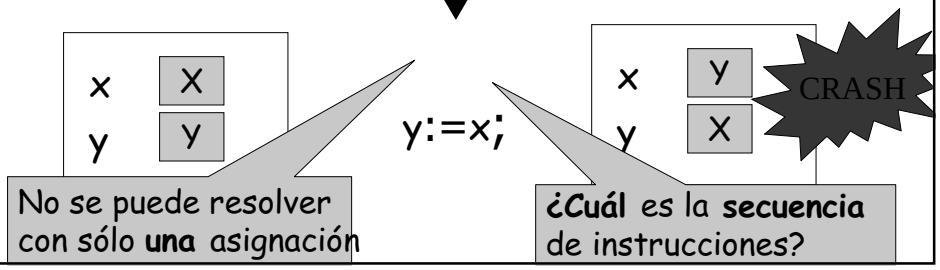
11

2.2. Composición secuencial

Ejemplo de <u>semántica</u> de la composición sec.

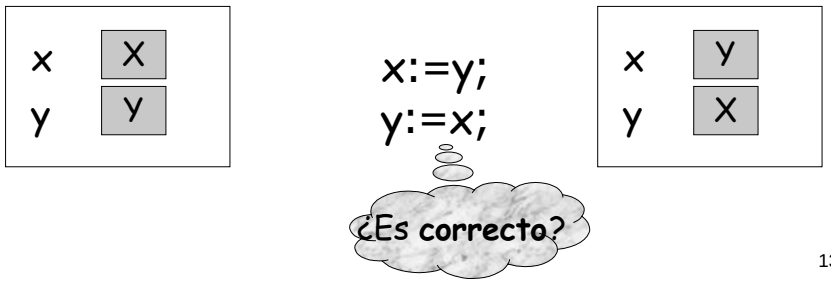
algoritmo intercambio(**DATOS** x, y: entero;
RESULTADOS x, y: entero)

P={x=X ∧ y=Y**}** intercambio **Q={**x=Y ∧ y=X**}**



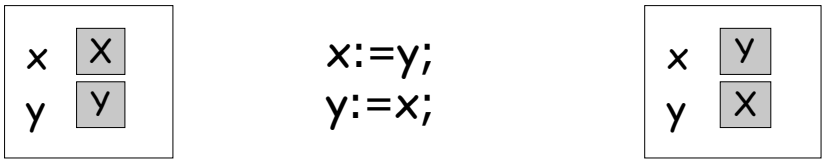
❑ Solución "directa"

P={x=X ∧ y=Y**}** intercambio **Q={**x=Y ∧ y=X**}**



13

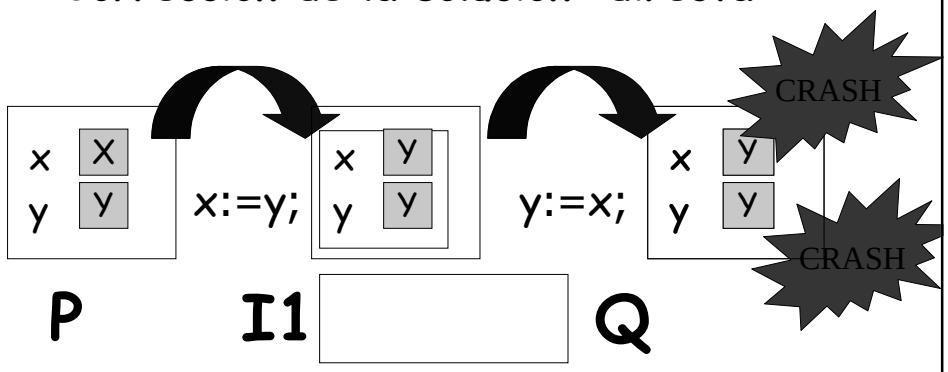
❑ Corrección de la solución "directa"



P **I1 ;I2** **Q**

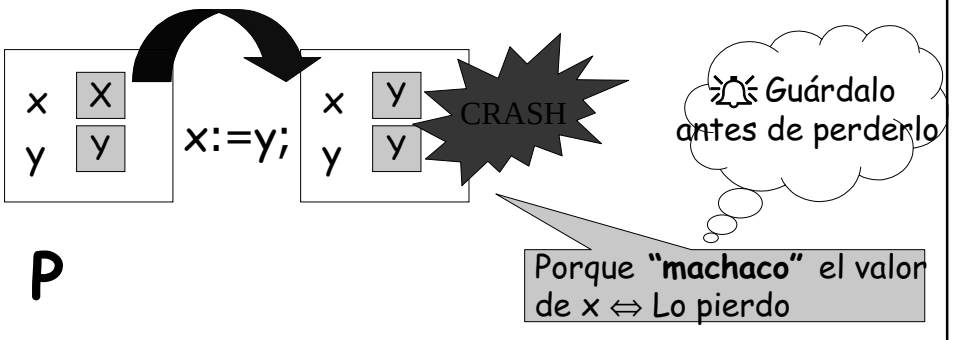
14

❑ Corrección de la solución "directa"



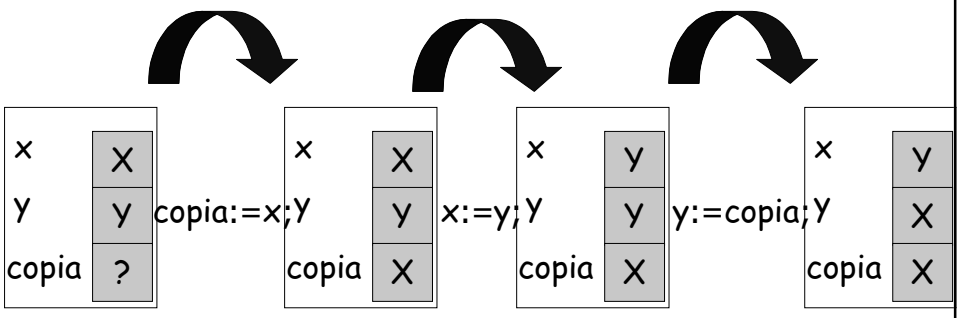
15

❑ Corrección de la solución "directa"



16

- **Construcción de la solución "correcta"**



17

2.2. Composición secuencial

- **Uso de la composición secuencial:**

  Problemas en los que **S** enuncia la **secuencia de acciones** que llevan de **P** a **Q**

Estrategia de diseño

- **Determinar objetivos parciales**
- **Plantear las instrucciones que permitan alcanzar los objetivos parciales**

18

2.2. Composición secuencial

- La composición secuencial **NO es conmutativa**

{x=3 ∧ y=Y}	**{x=3 ∧ y=Y}**
x:=0;	**y:=x+y;**
y:=x+y;	**x:=0;**
{x=0 ∧ y=Y}	**{x=0 ∧ y=Y+3}**

19

Dado un número de segundos inferior a un millón, calcular su equivalente en días, horas, minutos y segundos

algoritmo segundos
(**DATOS** n:entero;
RESULTADOS días,horas,min,seg: entero)

P= {0≤n<1.000.000}

¿Cómo comprobar si son **correctos**?

Q={n=86400 días + 3600 horas + 60 min + seg
∧ días≥0
∧ 0≤horas≤23
∧ 0≤min≤59
∧ 0≤seg≤59}

20

2.2. Composición secuencial

☞ **Estrategia de diseño: Serialización**

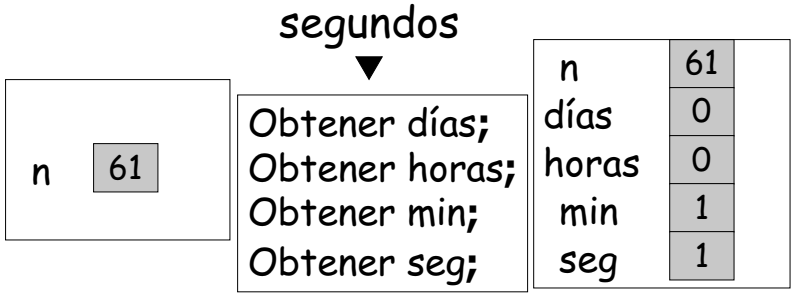
1.-Determinar objetivos parciales

2.- Determinar las instrucciones para alcanzarlos

21

2.2. Composición secuencial

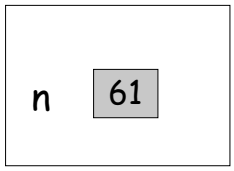
Serialización



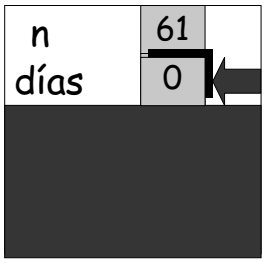
22

2.2. Composición secuencial

Serialización



días:=n div 86400;

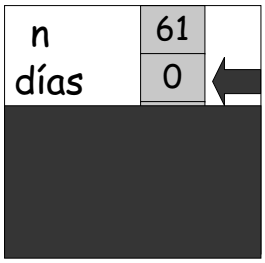


Q_1={n=86400 días + resto ∧
días ≥ 0 ∧ 0≤resto<86400}

23

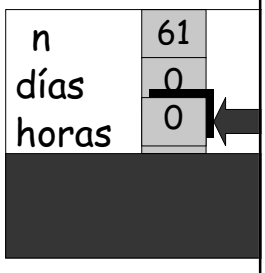
2.2. Composición secuencial

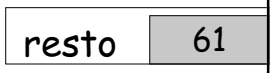
Serialización



resto:=n mod 86400;

horas:=resto div 3600;

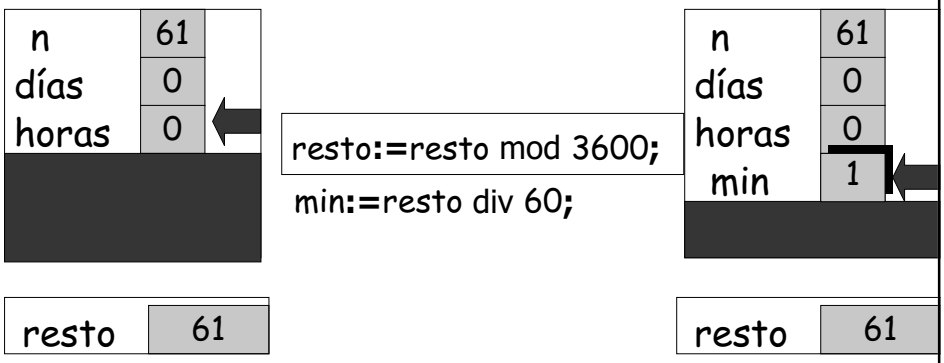




24

2.2. Composición secuencial

Serialización



25

2.2. Composición secuencial

Serialización

26

2.3. Condicionales: Análisis por casos

- **Sintaxis de la condicional**

```
opción
      caso_1: I1;
      ....
      caso_N : IN;
fopción
```

donde

- $caso_i$ es una expresión de tipo lógico
- **I1..IN** son instrucciones

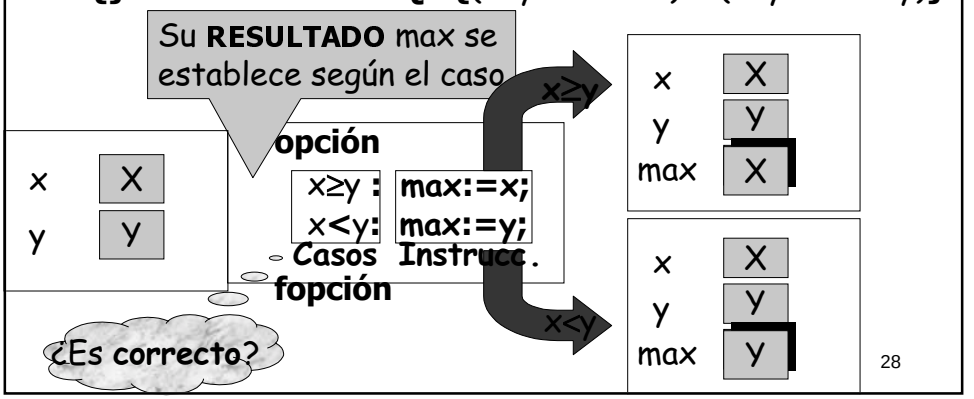
27

2.3. Condicionales: Análisis por casos

Ejemplo de semántica de la condicional

algoritmo máximo(**DATOS** x,y:entero;**RESULTADOS** max:entero)

P={} máximo **Q={**$(x \geq y \wedge max=x) \vee (x<y \wedge max=y)$**}**



28

2.3. Condicionales: Análisis por casos

- **Corrección de la Solución:**
  - ¿Se han controlado **todos** los casos?

    Sí, porque ó bien $x \geq y$ ó bien $x < y$

```
opción
  x≥y :  max:=x;
  x<y:  max:=y;
fopción
```


2.3. Condicionales: Análisis por casos

- **Corrección de la Solución:**
  - ¿Se han controlado **todos** los casos?
  - ¿Son casos **disjuntos**?

    Sí, porque $\neg(x \geq y \wedge x < y)$

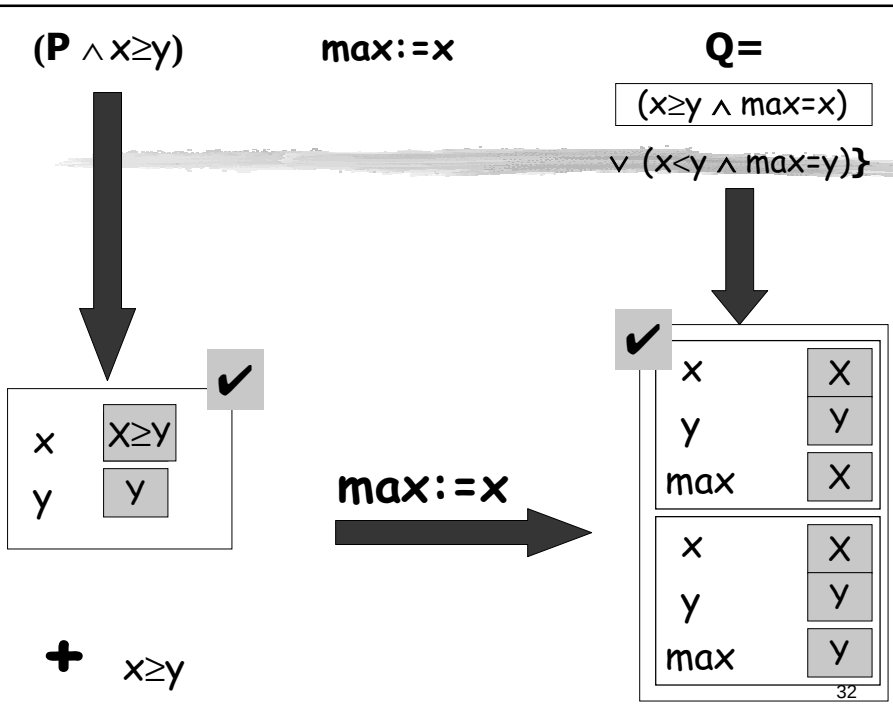
```
opción
  x≥y :  max:=x;
  x<y:  max:=y;
fopción
```

❑ Corrección de la Solución:

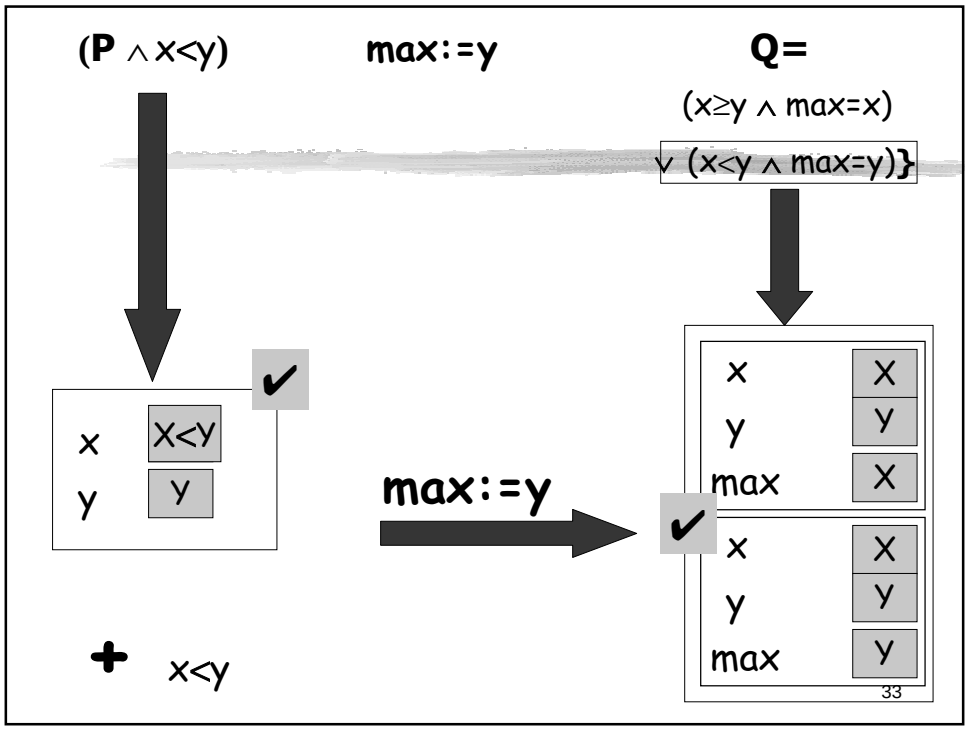
- ¿Se han controlado **todos** los casos?
- ¿Son casos **disjuntos**?
- Para cada caso, ¿**se ejecuta la instrucción correcta**?
  - Si $x \geq y$: **$(P \wedge x \geq y)$ max:=x Q**
  - Si $x < y$: **$(P \wedge x < y)$ max:=y Q**

opción
$x \geq y$: max:=x;
$x < y$: max:=y;
fopción

31



32



2.3. Condicionales: Análisis por casos

- **Uso de la condicional:**

  Problemas en los que existe una **partición** del dominio de los datos -**casos**. El análisis de cada caso determina la acción que lleva de **P** a **Q**

Estrategia de diseño

- **Establecer la partición de los datos -casos disjuntos**
- **Plantear las instrucciones para cada caso**

Otras formas de la condicional

```
si caso1 entonces  I1
        sino       I2;
fsi
```

≡

```
opción
     caso1:   I1;
     ¬caso1 : I2;
fopción
```

35

Otras formas de la condicional

```
si caso1 entonces  I1;
fsi
```

≡

```
opción
     caso1:   I1;
     ¬caso1 : vacía;
fopción
```

36

Otras formas de la condicional

```
opción
      caso1: I1;
      ....
      casoN : IN;
      en otro caso : IN+1;
fopción
```

≡

```
opción
      caso1: I1;
      ....
      casoN : IN;
      ¬ caso1 ∧ ... ∧ ¬ casoN : IN+1;
fopción
```

37

2.3. Condicionales: Análisis por casos

- **Condicional**

```
opción
      caso1: I1;
      ....
      casoN : IN;
fopción
```

en PASCAL "**if** " ~~ANIDADO~~
... y "**case**" en contadas ocasiones

38

2.3. Condicionales: Análisis por casos

- **Condicional más sencillo**

```
opción
      caso1  :  I1;
      ¬caso1 :  I2;
fopción
```

Ejemplo:

2 casos

```
opción
      caso1 x≥y  :   max:=x ;
      ¬caso1 x<y :   max:=y ;
fopción
```

39

2.3. Condicionales: Análisis por casos

- **Condicional más sencillo**

```
si caso1 entonces I1
         sino     I2;
fsi
```

Ejemplo:

2 casos

```
si   x≥y  entonces   max:=x
          sino       max:=y;
fsi
```

2.3. Condicionales: Análisis por casos

- **Condicional más sencillo** en PASCAL "**if**"

```
if caso1   then   I1
           else   I2;
```

Ejemplo:

```
if   x≥y   then   max:=x
           else   max:=y;
```

2.3. Condicionales: Análisis por casos

- **Condicional más sencillo**

```
opción
        caso1   :   I1;
        ¬caso1  :   vacía
fopción
```

```
si caso1 entonces I1
fsi
```

42

2.3. Condicionales: Análisis por casos

- **Condicional más sencillo** en PASCAL "**if**"

```
if caso1 then I1 ;
```

43

La siguiente tabla muestra los resultados result1 y result2 que se obtienen para los diferentes valores del dato ent:

ent	result1	result2
1	cierto	5
2	cierto	4
3	cierto	3
4	cierto	2
5	cierto	1
6	falso	3
7	falso	2
8	falso	1

Observando esta tabla se quiere diseñar un algoritmo:

- que obtenga los valores result1 y result2 dependiendo de los 8 posibles valores de ent
- que lo haga mediante un análisis de 2 casos <u>únicamente</u>

44

algoritmo tabla (**DATOS** ent: entero;
RESULTADOS result1: lógico;result2: entero)

P= {1≤ent≤8**}**

ent	result1	result2
1	cierto	5
2	cierto	4
3	cierto	3
4	cierto	2
5	cierto	1
6	falso	3
7	falso	2
8	falso	1

Los **RESULTADOS** se establecen según el caso

Q= { ((1≤ent≤5) → result1=**V** ∧ result2 = 5-ent+1)
∧ ((6≤ent≤8) → result1=**F** ∧ result2 = 8-ent+1)**}**

45

algoritmo tabla (**DATOS** ent: entero;
RESULTADOS result1: lógico;result2: entero)

P= {1≤ent≤8**}**

Q= {((1≤ent≤5) → result1=**V** ∧ result2 = 5-ent+1)
∧
((6≤ent≤8) → result1=**F** ∧ result2 = 8-ent+1)**}**

- **Diseño del algortimo** **2 casos**

opción
caso$_1$ 1≤ent≤5 **:** result1=**V ;** result2 = 5-ent+1 **;**
caso$_2$ 6≤ent≤8 **:** result1=**F ;** result2 = 8-ent+1 ;
fopción

¡ j !

caso$_2$ <> ¬ **caso**$_1$

46

algoritmo tabla (**DATOS** ent: entero;
RESULTADOS result1: lógico;result2: entero)

P= {1≤ent≤8**}**

Q= {((1≤ent≤5) → result1=V ∧ result2 = 5-ent+1)
∧
((6≤ent≤8) → result1=F ∧ result2 = 8-ent+1)**}**

❑ Diseño del algortimo

opción
1≤ent≤5 : result1=V ;
result2 = 5-ent+1 ;
6≤ent≤8 : result1=F ;
result2 = 8-ent+1 ;
fopción

si 1≤ent≤5
entonces result1=V ;
result2 = 5-ent+1 ;
sino result1=F ;
result2 = 8-ent+1 ;
fsi

47

Algoritmo VERSUS Programa

algoritmo tabla(**DATOS** ent : entero;
RESULTADOS result1 :lógico;
result2 :entero);

P={1≤ent≤8**}**

si 1≤ent≤5
entonces result1=V ;
result2 = 5-ent+1;
sino result1=F ;
result2 = 8-ent+1;
fsi

Q={((1≤ent≤5)
→ result1=V ∧ result2=5-ent+1)
∧((6≤ent≤8)
→ result1=F ∧result2=8-ent+1)**}**

falgoritmo

```
PROGRAM tabla (INPUT,OUTPUT);
VAR ent,result2:INTEGER;
    result1:BOOLEAN;
BEGIN
 WRITELN('entrada entre 1 y 8');
 READLN(ent);
IF (ent>=1) and (ent<=5)
THEN BEGIN result1:=true;
           result2:=6-ent;
     END
ELSE BEGIN result1:=false;
           result2:=9-ent;
     END;
WRITE('result1=',result1:6);
WRITELN(';result2=',result2:2);

END.
```

48

```
$ gpc -o tabla   tabla.p
```

```
$ tabla
entrada entre 1 y 8
1
result1=  true;result2= 5
```

```
$ tabla
entrada entre 1 y 8
6
result1= false;result2= 3
```

```
$ tabla
entrada entre 1 y 8
-1
result1= false;result2= 10
```

P={1≤ent≤8}

```
PROGRAM tabla (INPUT,OUTPUT);
VAR ent,result2:INTEGER;
    result1:BOOLEAN;
BEGIN
 WRITELN('entrada entre 1 y 8');
 READLN(ent);
 IF (ent>=1) and (ent<=5)
 THEN BEGIN result1:=true;
            result2:=6-ent;
      END
 ELSE BEGIN result1:=false;
            result2:=9-ent;
      END;
 WRITE('result1=',result1:6);
 WRITELN(';result2=',result2:2);
END.
```

caso$_2$ <> ¬ caso$_1$

49

SOLUCIÓN PASCAL: "**if** " ANIDADO

```
PROGRAM tabla (INPUT,OUTPUT);
VAR ent,result2:INTEGER;
    result1:BOOLEAN;
BEGIN
 WRITELN('entrada entre 1 y 8');
 READLN(ent);
 IF (ent>=1) and (ent<=5)
 THEN BEGIN result1:=true;
            result2:=6-ent;
 END
 ELSE BEGIN result1:=false;
            result2:=9-ent;
 END;
 WRITE('result1=',result1:6);
 WRITELN(';result2=',result2:2);
END.
```

P={1≤ent≤8}

```
IF (ent<1) or (ent>8)
THEN WRITELN('Datos no validos')
ELSE BEGIN
      IF ent<=5
      THEN BEGIN result1:=true;
                 result2:=6-ent;
           END
      ELSE BEGIN result1:=false;
                 result2:=9-ent;
           END;
      END;
```

50

Calcular el máximo y mínimo de tres números enteros, a b y c, distintos entre sí

algoritmo max_min(**DATOS** a, b, c : entero;
RESULTADOS max,min: entero**)**

P= {a≠b≠c**}**

Q= {max=máximo(a,b,c) ∧ min=mínimo(a,b,c)**}**

donde máximo y mínimo son dos funciones de especificación que se dejan propuestas al alumno

51

Versión 1 :

Estrategia de diseño: Análisis por casos

Casos en los que a es el mínimo: a<b<c a<c<b
Casos en los que b es el mínimo: b<a<c b<c<a
Casos en los que c es el mínimo: c<a<b c<b<a

*Partición en **6** casos disjuntos

52

Versión 1 :

Estrategia de diseño: Análisis por casos

Casos en los que a es el mínimo:
a<b<c:min=a; max=c
a<c<b:min=a; max=b
Casos en los que b es el mínimo:
b<a<c:min=b; max=c
b<c<a:min=b; max=a
Casos en los que c es el mínimo:
c<a<b:min=c; max=b
c<b<a:min=c; max=a

*Partición en **6** casos disjuntos

*Instrucciones para cada caso

53

Versión 1 :

opción

a<b<c:	min=a; max=c
a<c<b:	min=a; max=b
b<a<c:	min=b; max=c
b<c<a:	min=b; max=a
c<a<b:	min=c; max=b
c<b<a:	min=c; max=a

fopción

54

Versión 1 : Traducción a Pascal

if (a<b**) and (**b<c**) then begin**	min=a; max=c	**end;**
if (a<c**) and (**c<b**) then begin**	min=a; max=b	**end;**
b<a<c:	min=b; max=c	
b<c<a:	min=b; max=a	
c<a<b:	min=c; max=b	
if (c<b**) and (**b<a**) then begin**	min=c; max=a	**end;**

55

Versión 1 — Traducción a Pascal mejorada

opción

a<b<c: min=a; max=c

if (a<b**) and (**b<c**)**

then begin min=a; max=c**; end**

else (* a>b **or** b>c ***)**

Versión 1

Traducción a Pascal mejorada

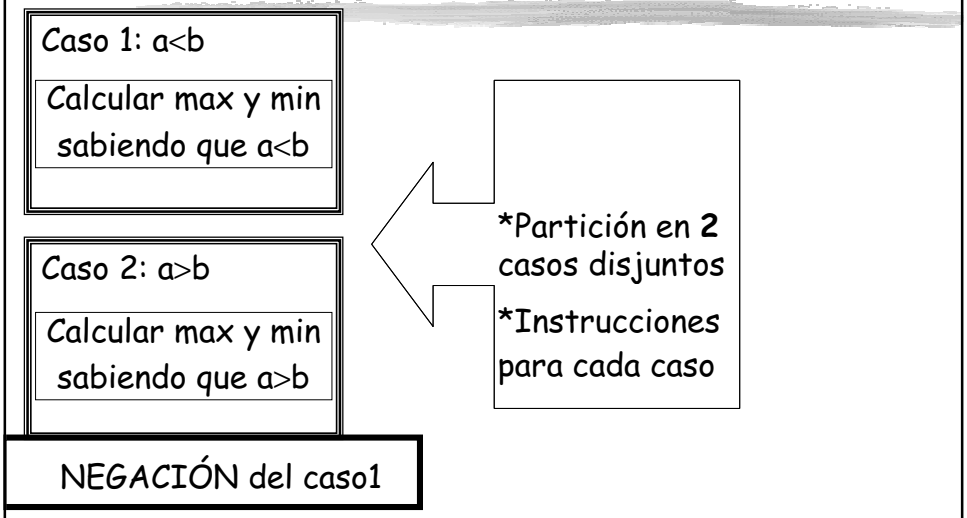
```
if (a<b) and (b<c)
then begin min=a; max=c; end
else (* a>b or b>c *)
     if  a>b then  if b>c
                   then begin min=c; max=a; end
                   else begin (* c>b and a>b *)
                              min=b;
                              if a>c then max=a else max=c;
                        end
```

57

```
if (a<b) and (b<c)
then begin min=a; max=c; end
else (* a>b or  b>c *)
if  a>b then  if  b>c
              then begin min=c; max=a; end
              else begin (* c>b and a>b *)
                         min=b;
                         if a>c then max=a else max=c;
                   end
     else begin (* a<b and c<b *)
             max=b ;
              if a<c then min=a; else min=c ;
          end;
```

Versión 2 :

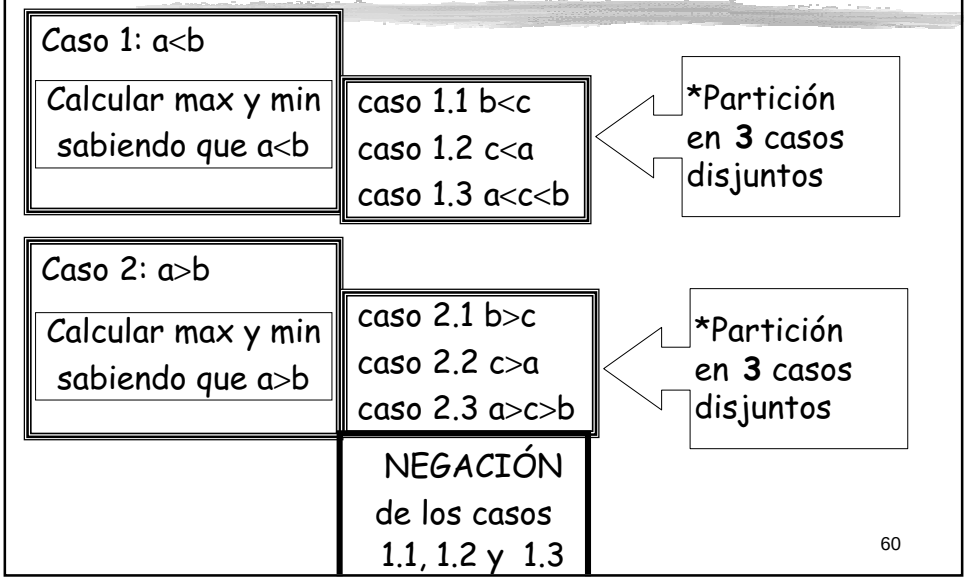
Estrategia : Análisis por casos **ANIDADO**



59

Versión 2 :

Estrategia : Análisis por casos **ANIDADO**



60

Versión 2 :

Estrategia : Análisis por casos **ANIDADO**

Caso 1: a<b Calcular max y min sabiendo que a<b	caso 1.1 b<c caso 1.2 c<a caso 1.3 a<c<b	: min=a; max=c : min=c; max=b : min=a; max=b
Caso 2: a>b Calcular max y min sabiendo que a>b	caso 2.1 b>c caso 2.2 c>a caso 2.3 a>c>b	: min=c; max=a : min=b; max=c : min=b; max=a

NEGACIÓN de las instrucciones para los casos 1.1, 1.2 y 1.3

61

Versión 2

```
opción
   a<b :     opción
                  b<c    : min := a ; max := c ;
                  c<a    : min := c ; max := b ;
                  a<c<b  : min := a ; max := b ;
             fopción
   a>b :     opción
                  b>c    : min := c ; max := a ;
                  c>a    : min := b ; max := c ;
                  a>c>b  : min := b ; max := a ;
             fopción
fopción
```

62

Versión 2

Traducción a Pascal

```
if  a<b then
        if b<c then begin max:=c;min:=a; end
                else begin (*b>c*)
                        max:=b;
                        if a>c then min:=c else min:=a ;
                     end
    else

        lo anterior PERO justo al contrario
```

63

Versión 3 :

Estrategia : Serialización y Análisis por casos

```
Obtener min/max de a y b ;
si a>b entonces max:=a
        sino      max:=b;
fsi
Obtener min/max de min/max y c ;
si c>max entonces max:=c
        sino
        si c<min entonces min:=c ;
fsi
```

Análisis de **2** casos

Instrucciones para obtener los **objetivos parciales**

Análisis de **2** casos

64

Versión 3

```
si a>b        entonces max:=a
              sino     max:=b;
fsi
(*max>min*)
si c>max      entonces max:=c
              sino     si c<min entonces min:=c ;
fsi
```

65

Versión 3 — Traducción a Pascal

```
if a>b      then    max:=a
            else    max:=b;

(*max>min*)
if c>max    then    max:=c
            else    if c<min then min:=c ;
```

66