

Bloque 1. Conceptos y técnicas básicas en programación



1. Introducción
2. Datos y expresiones. Especificación de algoritmos
3. Estructuras algorítmicas básicas
4. Iteración y recursión
5. Iteración y recursión sobre secuencias
6. Iteración y recursión sobre tablas

Notas:



1. Introducción
2. Datos y expresiones. Especificación de algoritmos
3. Estructuras algorítmicas básicas
 - La asignación. Composición secuencial de instrucciones. La instrucción alternativa simple y múltiple. Alternativa exclusiva y general.
4. Iteración y recursión
5. Iteración y recursión sobre secuencias
6. Iteración y recursión sobre tablas

3.1. La asignación

La instrucción de asignación consiste en dar valor a una variable

Sintaxis:

Variable := expresión

- primero se evalúa la expresión
- después el resultado se guarda en la variable
- el valor anterior de la variable se pierde
 - aunque se puede usar en la expresión

El tipo de la expresión debe ser compatible con el tipo de la variable

Corrección de la asignación: Ejemplo

Podemos averiguar la corrección de un algoritmo que dispone de una asignación, comprobando la precondition y postcondición

```

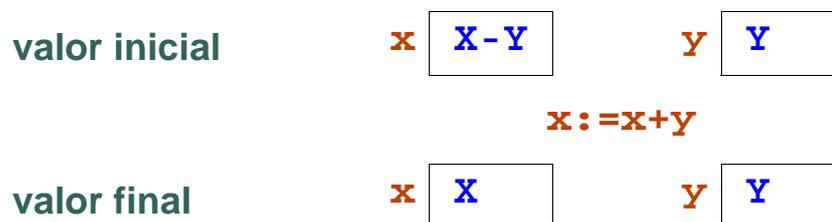
algoritmo ejemplo
  // debemos encontrar la expresión E que hace
  // correcto el algoritmo
  var
    entero x,y;
  fvar
    {Pre: x+y=X, y=Y}
    x:=E;
    {Post: x=X, y=Y}
falgoritmo
  
```

Corrección de la asignación: solución al ejemplo

E es la expresión $x+y$

Explicación:

- la postcondición es $\{x=X, y=Y\}$
- si sustituimos x por $x+y$ en la postcondición obtenemos $\{x+y=X, y=Y\}$
 - que es justamente la precondition



Corrección de la asignación

Llamamos R_E^x a la sustitución en el predicado R de toda aparición de la variable x por la expresión E

Axioma de la asignación

- para toda variable x y toda expresión válida E del mismo tipo y todo predicado R , la especificación siguiente es correcta

$$\begin{array}{l} \{R_E^x\} \\ \quad x := E; \\ \{R\} \end{array}$$

Regla de inferencia de la asignación

Combinando el axioma de la asignación con la primera regla de consecuencia de una especificación (ver capítulo 2), se obtiene

- Para que esta especificación sea correcta

```

{P}
  x := E;
{Q}
```

- se debe cumplir que $P \Rightarrow Q_E^x$

3.2. Composición secuencial de instrucciones

Una de las estructuras algorítmicas básicas es la composición secuencial

- ejecutar un algoritmo a continuación de otro
- el orden en que se ejecutan influye en el resultado

Sintaxis

```
algoritmo 1;
algoritmo 2
```

o

```
algoritmo 1; algoritmo 2
```

Composición secuencial de asignaciones

Una composición de dos asignaciones

```

var
  tipo1 x1;
  tipo2 x2;
fvar
{Pre:P}
  x1:=E1;
  x2:=E2;
{Post:Q}

```

Es correcta si $P \Rightarrow Q_{E2}^{x2} \)_{E1}^{x1}$

Si son n expresiones, $P \Rightarrow Q_{En}^{xn} \)_{En-1}^{xn-1} \ \dots \)_{E1}^{x1}$

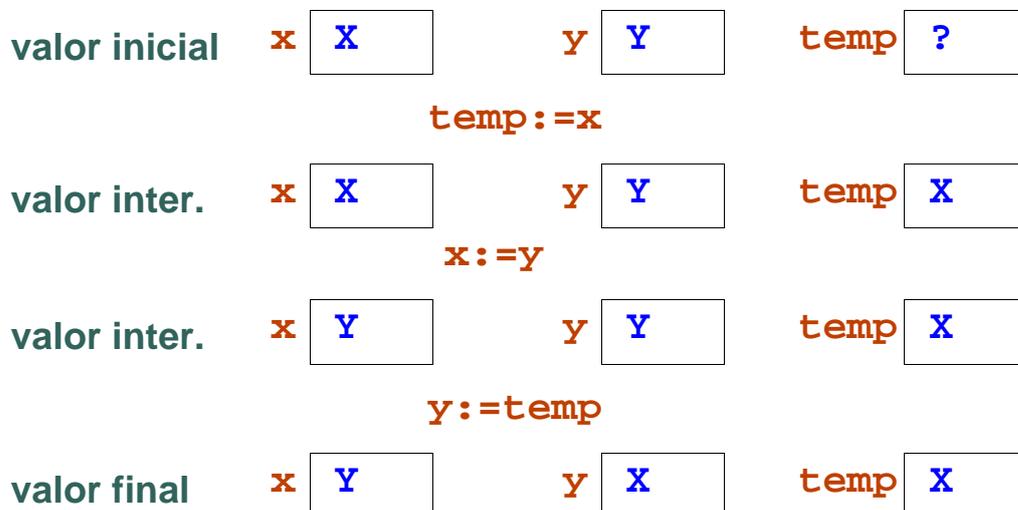
Ejemplos

```

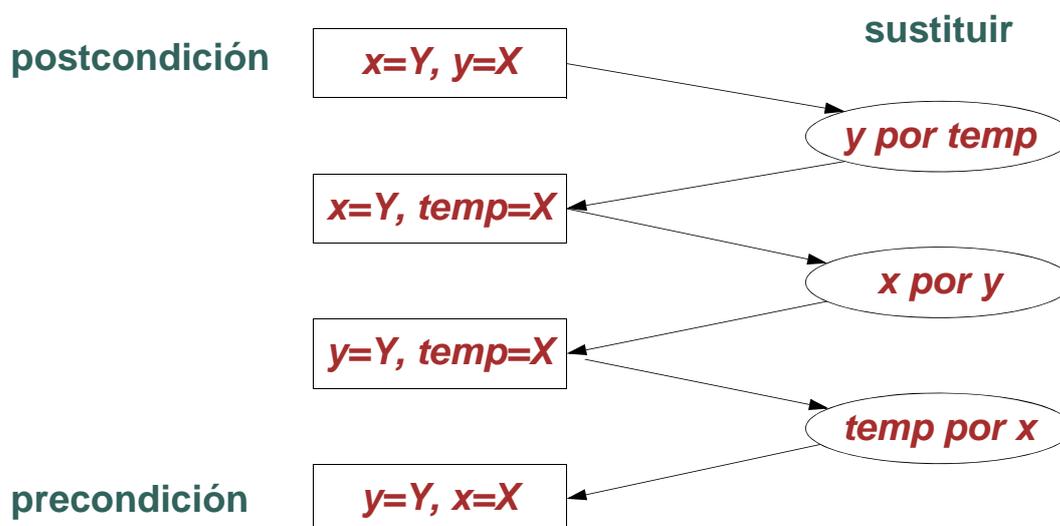
algoritmo intercambiar
// debemos intercambiar los valores de x y de y
var
  entero x,y;
fvar
{Pre: x=X, y=Y}
  var
    entero temp;
  fvar
    temp:=x;
    x:=y;
    y:=temp;
  {Post: x=Y, y=X}
falgoritmo

```

Traza de intercambiar



Corrección de intercambiar



Ejemplo: clase Circulo

| Circulo |
|--|
| real centroX real centroY real radio |
| Circulo(real x, real y, real r) real centroX() real centroY() real radio() real area() dibuja() |

Especificación de los métodos

```
método Circulo(real x, real y, real r)
    {Pre: x=X, y=Y, r=R, r>=0}
    constructor
    {Post: centroX=X & centroY=Y & radio=R}
```

fmétodo

```
método centroX() retorna real
    {Pre:}
    obtener la coordenada x del centro
    {Post: valor retornado=centroX}
```

fmétodo

// centroY y radio son similares

Especificación de los métodos (cont.)

```
método area() retorna real
  {Pre:}
  cálculo del área
  {Post: valor retornado= $\pi \cdot \text{radio}^2$ }
fmétodo
```

```
método dibuja(
  {Pre:}
  dibuja el círculo en la pantalla
  {Post: el círculo se ha dibujado en la pantalla}
fmétodo
```

Diseño de los métodos

```
método Circulo(real x, real y, real r)
  {Pre: x=X, y=Y, r=R, R>=0}
  centroX:=x;
  centroY:=y;
  radio:=r;
  {Post: centroX=X & centroY=Y & radio=R}
fmétodo
```

```
método centroX() retorna real
  {Pre:}
  retorna centroX;
  {Post: valor retornado=centroX}
fmétodo
```

```
// centroY y radio son similares
```

Diseño de los métodos (cont.)

```
método area() retorna real
  {Pre:}
  retorna  $\pi \cdot \text{radio}^2$ ;
  {Post: valor retornado= $\pi \cdot \text{radio}^2$ }
fmétodo

// el método dibuja es complejo y requiere
// instrucciones que aún no conocemos
```

3.3. La instrucción alternativa

La instrucción alternativa, o condicional, nos permite ejecutar unas instrucciones u otras en función de:

- **una condición booleana:** alternativa simple
- **múltiples condiciones booleanas:** alternativa múltiple
 - alternativa **exclusiva:** las condiciones son disjuntas (es decir, sólo una puede ser cierta)
 - alternativa **general:** pueden cumplirse varias de las condiciones

Sintaxis de la alternativa simple

Ejecución condicional de instrucciones

```

si condición entonces
    instrucciones
fsi
  
```

Ejecución condicional de una de entre dos alternativas

```

si condición entonces
    instrucciones
si no
    instrucciones
fsi
  
```

Ejemplo 1: valor absoluto

```

método valorAbsoluto(real x) retorna real
  {Pre:}
    si x>=0 entonces
      retorna x;
    si no
      retorna -x;
    fsi
  {Post: valor retornado=|x|}
fmétodo
  
```

Ejemplo 2: máximo de dos enteros

```
método maximo(real x,y) retorna real
  {Pre:}
  si x>=y entonces
    retorna x;
  si no
    retorna y;
  fsi
  {Post: valor retornado=máximo(x,y)}
fmétodo
```

Sintaxis de la alternativa múltiple

La condición depende de múltiples predicados booleanos

```
si
  condicion1 -> instrucciones;
  condicion2 -> instrucciones;
  condicion3 -> instrucciones;
fsi
```

Las condiciones se evalúan en el orden en que aparecen

- cuando una es cierta, se ejecutan sus instrucciones y se termina la instrucción condicional

Cada conjunto de instrucciones debe conducir a la postcondición

Recomendación: Deben cubrirse todos los casos (al menos una de las condiciones debe ser cierta)

3.4. Alternativa exclusiva y general

Alternativa exclusiva

- Las condiciones son disjuntas (es decir si una es cierta, ninguna de las otras puede ser cierta)

Alternativa general

- Las condiciones no tienen por qué ser disjuntas
- el orden en que se escriben influye en el resultado

La alternativa simple es siempre exclusiva

Ejemplo: ordenación de tres números

Especificación:

```

var
  p,q,s : real
fvar
método ordenar3Numeros (real x,y,z)
  {Pre:}
  ordenar
  {Post: p<=q<=s, (p,q,s) es permutación de (x,y,z)}
fmétodo
  
```

Ejemplo: alternativa general

```

método ordenar3Numeros (real x,y,z)
  {Pre:}
  si
    (x<=y)&(y<=z) -> p:=x; q:=y; s:=z;
    (x<=z)&(z<=y) -> p:=x; q:=z; s:=y;
    (y<=x)&(x<=z) -> p:=y; q:=x; s:=z;
    (y<=z)&(z<=x) -> p:=y; q:=z; s:=x;
    (z<=x)&(x<=y) -> p:=z; q:=x; s:=y;
    (z<=y)&(y<=x) -> p:=z; q:=y; s:=x;
  fsi
  {Post: p<=q<=s, (p,q,s) es permutación de (x,y,z)}
fmétodo

```

Observar que, por ejemplo, si $x=y=z$, se cumplen todas las condiciones

Ejemplo: alternativa exclusiva

```

método ordenar3Numeros (real x,y,z)
  {Pre:}
  si
    (x<y)&(z<x) -> p:=z; q:=x; s:=y;
    (x<y)&(x<=z)&(z<y) -> p:=x; q:=z; s:=y;
    (x<y)&(y<=z) -> p:=x; q:=y; s:=z;
    (y<=x)&(z<y) -> p:=z; q:=y; s:=x;
    (y<=x)&(y<=z)&(z<x) -> p:=y; q:=z; s:=x;
    (y<=x)&(x<=z) -> p:=y; q:=x; s:=z;
  fsi
  {Post: p<=q<=s, (p,q,s) es permutación de (x,y,z)}
fmétodo

```

Ejemplo: dos alternativas exclusivas

```

método ordenar3Numeros (real x,y,z)
  si x<y entonces
    si
      (z<x)           -> p:=z; q:=x; s:=y;
      (x<=z)&(z<y)    -> p:=x; q:=z; s:=y;
      (y<=z)         -> p:=x; q:=y; s:=z;
    fsi
  si no
    si
      (z<y)           -> p:=z; q:=y; s:=x;
      (y<=z)&(z<x)    -> p:=y; q:=z; s:=x;
      (x<=z)         -> p:=y; q:=x; s:=z;
    fsi
  fsi
fmétodo

```

Ejemplo: tres alternativas

Método de la burbuja

```

método ordenar3Numeros (real x,y,z)
  {Pre:}
  p:=x; q:=y; s:=z;
  si (p>q) entonces
    intercambiar p y q;
  fsi
  si (p>s) entonces
    intercambiar p y s;
  fsi
  si (q>s) entonces
    intercambiar q y s;
  fsi
  {Post: p<=q<=s, (p,q,s) es permutación de (x,y,z)}
fmétodo

```