Bloque 1. Conceptos y técnicas básicas en programación

UNIVERSIDAD DE CANTABRIA

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:

UNIVERSIDAD DE CANTABRIA

1. Introducción

2. Datos y expresiones. Especificación de algoritmos

3. Estructuras algorítmicas básicas

4. Iteración y recursión

- Diseño iterativo. Instrucciones de bucle. Invariantes y cotas. Fases de un diseño iterativo. Ejemplos. Recursión. Ejemplos. Corrección de la implementación recursiva. Fases del diseño recursivo.

5. Iteración y recursión sobre secuencias

6. Iteración y recursión sobre tablas

Diseño iterativo

Se trata de indicar al computador cómo se calcula la función especificada *repitiendo* muchas veces un conjunto de instrucciones

La sintaxis de un algoritmo iterativo es:

```
mientras B hacer
    S;
fmientras
```

- donde `B` es un predicado y `S` un conjunto de instrucciones

La estructura completa se llama *bucle*, `B` se llama *condición de continuación* y `S` *cuerpo del bucle*

Ejemplo

Vamos a escribir un método para obtener el valor del *logaritmo* de `y=1+x` de acuerdo con el siguiente desarrollo en serie

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots + (-1)^{n-1}\frac{x^n}{n}, \quad -1 < x \leq 1$$

Especificación

```
método logaritmo (real y, entero n) retorna real
   {Pre: 0<y<=2, n>0}

      desarrollo en serie

   {Post: x=y-1, valor retornado = }
fmétodo
```

$$\sum_{i=1}^{n} (-1)^{i-1} \frac{x^i}{i}$$

Diseño

```
método logaritmo (real y, entero n) retorna real
   {Pre: 0<y<=2, n>0}
      var
         real x:=y-1; real numerador:=x; real log:=0;
         entero signo:=1; entero i:=1;
      fvar
      mientras i<=n hacer
         log:=log+signo*numerador/i;
         numerador:=numerador*x;
         signo:=-signo;
         i++;
      fmientras
      retorna log;
   {Post: x=y-1, valor retornado = }
fmétodo
```

$$\sum_{i=1}^{n} (-1)^{i-1} \frac{x^i}{i}$$

Traza de un proceso iterativo

Estado al comenzar el bucle, en cada una de sus sucesivas iteraciones

y	x	i	numerador	signo	log
1.2	0.2	1	0.2	1	0
1.2	0.2	2	0.04	-1	0.19999
1.2	0.2	3	0.008	1	0.17999
1.2	0.2	4	0.0016	-1	0.18266

Invariantes

Es una descripción del estado de las variables al comenzar la iteración

- se mantiene inalterada después de cada iteración

Invariante del ejemplo

- `log` contiene la suma de los términos de la serie hasta `i-1`
- `i` es el término que vamos a tratar, `1<=i<=n+1`
- `signo` es -1^{i-1}
- `numerador` es x^i

Cuando nos salimos del bucle:

- `i` vale `n+1`, y `log` contiene la suma de la serie hasta `i-1=n`

Relación del invariante con la estructura del bucle

```
inicializaciones;
{se cumple el Invariante I}
mientras B hacer
    {se cumplen I y B}
    S;
    {se cumple I}
fmientras
{se cumple I y no B}
{se cumple la Postcondición}
```


Fases de un diseño iterativo

Diseñar el invariante

- para que indique el resultado que se obtendrá a cada paso del bucle

Determinar las inicializaciones para que se cumpla el invariante

Determinar la condición de continuación, que nos permite saber cuándo abandonamos el bucle

Determinar el cuerpo del bucle

- comprobar que se sigue cumpliendo el invariante cada vez

Verificar que el bucle termina

Razonar sobre el estado obtenido a la terminación

Terminación del bucle

Para verificar que el bucle termina puede plantearse la función de *cota*

- **indica la "distancia" máxima hasta la finalización del bucle**
- **debe ir reduciéndose con cada iteración, hasta llegar a cero**

En el ejemplo de la suma de una serie, la función de cota es el número de iteraciones que faltan: `n+1-i`

- **a medida que `i` avanza, la función de cota se reduce**
- **cuando `i` llega al valor `n+1` la cota es cero y el bucle termina**

Ejercicios propuestos

1. Calcular la potencia entera de un número real x: x^n

2. Calcular el seno de un ángulo x usando el siguiente desarrollo en serie:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}, \quad x \in R$$

Sintaxis alternativas: bucle con condición de permanencia al final

En ocasiones interesa evaluar la condición de permanencia al finalizar el bucle

- cuando se requiere para su evaluación haber ejecutado las instrucciones del bucle
- sintaxis

```
hacer
    instrucciones;
mientras condición;
```


Sintaxis alternativas: bucle con variable de control

En otras ocasiones hay bucles en que el número de veces en que se hace el bucle se cuenta mediante una variable de control

- sintaxis:

```
para i=0 hasta i=100 hacer
    instrucciones
fpara
```

O con un paso diferente a la unidad

- sintaxis:

```
para i=0 hasta i=100 paso 2 hacer
    instrucciones
fpara
```

Ejemplo

Algoritmo iterativo para el cálculo del *factorial* de un número natural

$$n! = 1 \quad , n = 0$$

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n, \quad n \geq 1$$

Especificación

```
método factorial (n:entero) retorna entero
   {Pre: n>=0}
      cálculo del factorial
   {Post: valor retornado=n!}
fmétodo
```


Diseno iterativo del método factorial

Usaremos el bucle con variable de control

Invariante:

- `i` es el paso a realizar; su rango de valores es: `1<=i<=n+1`
- `f` tiene el resultado hasta el paso i-1, `f=(i-1)!`,

Inicializaciones:

- `i:=1`
- `f:=1`

Cuerpo del bucle:

- `f:=f*i`
- `i++;`

Estructura final

```
método factorial (n:entero) retorna entero
   {Pre: n>=0}
      var
        entero f:=1; entero i:=1;
      fvar
      {Invariante I: f=(i-1)!, 1<=i<=n+1}
      mientras i<=n hacer
         {se cumple I e 1<=i<=n}
         f:= f*i;
         i++;
      fpara
      {se cumple I e i=n+1}
      retorna f;
   {Post: valor retornado=n!}
fmétodo
```


Estructura final usando el bucle con variable de control

```
método factorial (n:entero) retorna entero
   {Pre: n>=0}
      var
        entero f:=1;
      fvar
      para i=1 hasta i=n hacer
         {Invariante: f=(i-1)!, 1<=i<=n+1}
         f:= f*i;
      fpara
      retorna f;
   {Post: valor retornado=n!}
fmétodo
```

Recursión

Muchos algoritmos iterativos pueden resolverse también con un algoritmo *recursivo*

- el algoritmo se invoca a sí mismo
- en ocasiones es más natural
- en otras ocasiones es más engorroso

El *diseño recursivo* consiste en diseñar el algoritmo mediante una estructura alternativa de dos ramas

- ***caso directo*: resuelve los casos sencillos**
- ***caso recursivo*: contiene alguna llamada al propio algoritmo que estamos diseñando**

Ejemplos

Definición iterativa para el cálculo del *factorial* de un número natural

$$n! = 1 \quad , n = 0$$

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n, \quad n \geq 1$$

Definición recursiva para el cálculo del *factorial* de un número natural

$$n! = 1 \quad , n = 0$$

$$n! = n \cdot (n-1)!, \quad n \geq 1$$

La definición es correcta pues el número de recursiones es finito

Ejemplo: Diseño del algoritmo

```
método factorial (n:entero) retorna entero
   {Pre: n>=0}
      si n<=1 entonces
         // caso directo
         retorna 1;
      si no
         // caso recursivo
         retorna n*factorial(n-1);
      fsi
   {Post: valor retornado=n!}
fmétodo
```


Corrección de la implementación recursiva

***Caso directo*: comprobar que este caso conduce a un resultado correcto**

- **para `n=0` y `n=1`, el resultado obtenido es `1`**

***Caso recursivo*: comprobar por inducción que este caso también conduce a un resultado correcto**

- **para `n>=2`**

$$n! = n \cdot (n-1)!$$

$$n! = n \cdot (n-1) \cdot (n-2)!$$

$$\ldots$$

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$$

Corrección de la implementación recursiva

Terminación: comprobar la función de cota, al igual que en el diseño iterativo

- la cantidad de recursiones es el parámetro `n`, que toma valores sucesivamente menores hasta llegar a `1`

Fases del diseño recursivo

Obtener una definición recursiva de la función a implementar a partir de la especificación

- Establecer caso directo
- Establecer caso recursivo

Diseñar el algoritmo con una composición alternativa

- Diseñar el caso directo
- Diseñar el caso recursivo usando la hipótesis de inducción

Argumentar sobre la terminación del algoritmo

- Establecer la función de cota

Ejemplos propuestos

Especificar y diseñar un algoritmo recursivo que escriba las cifras de un número decimal `d` en cualquier base `b`

Diseñar un algoritmo recursivo para calcular el producto de números naturales usando sumas/restas, de acuerdo con esta definición

$$producto(x, 0) = 0$$

$$producto(x, y) = producto(x, y-1) + x, \quad y > 0$$

Ejemplos propuestos (cont.)

Diseñar un algoritmo recursivo para calcular el producto de dos números naturales usando sumas/restas y multiplicaciones/ divisiones por 2 (*multiplicación rápida*)

$$producto(x, 0) = 0$$

$$producto(x, y) = producto(x, y-1) + x, \; y > 0 \text{ es impar}$$

$$producto(x, y) = producto(2x, y/2), \; y > 0 \text{ es par}$$

Ejemplos propuestos (cont.)

Obtener el máximo común divisor *mcd* de dos números naturales (*x*,*y*) usando sólo sumas y restas

$$x > y \Rightarrow mcd = mcd(x - y, y)$$
$$y > x \Rightarrow mcd = mcd(x, y - x)$$
$$x = y \Rightarrow mcd = x$$

Obtener el máximo común divisor de dos números naturales usando sumas y restas y también los operadores `div` y `mod` (algoritmo de *Euclides*)

$$y = 0 \Rightarrow mcd = x$$
$$y > 0 \Rightarrow mcd = mcd(y, x \bmod y)$$

Consideraciones sobre los datos

Datos compartidos por todas las invocaciones del algoritmo

- **atributos del objeto**
- **estado de otros objetos externos**

Datos para los que cada invocación tiene una copia posiblemente distinta

- **variables locales (internas) del algoritmo**
- **parámetros**
- **valor de retorno del método**

Ejemplo: Diseño recursivo de la función logaritmo

Definición iterativa

$$\ln(1+x,n) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots + (-1)^{n-1}\frac{x^n}{n}, \quad -1 < x \leq 1$$

Definición recursiva

$$\ln(1+x,i,n) = (-1)^{i-1}\frac{x^i}{i} + \ln(1+x,i+1,n), -1 < x \leq 1$$

$$\ln(1+x,i,n) = (-1)^{i-1}\frac{x^i}{i}, \quad i = n$$

$$\ln(1+x,n) = \ln(1+x,1,n)$$

Especificación

```
método logaritmo (real y, entero i, n) retorna real
   {Pre: 0<y<=2, 1<=i<=n, n>0}

      desarrollo en serie

   {Post: x=y-1, valor retornado = } 
fmétodo
```

$$\sum_{j=i}^{n} (-1)^{j-1}\frac{x^j}{j}$$

Ejemplo: Diseño recursivo de la función logaritmo (incorrecto)

```
método logaritmo (real y, entero i, n) retorna real
   {Pre: 0<y<=2, 1<=i<=n, n>0}
      var
         real x:=y-1;  real término;
         real numerador:=x; entero signo:=1;
      fvar
      numerador:=numerador*x; signo:=-signo;
      término:=signo*numerador/i;
      si i==n entonces
         retorna término;
      si no
         retorna término+logaritmo(y,i+1,n);
      fsi;
   {Post: igual que antes}
fmétodo
```


Ejemplo (cont.)

El diseño anterior es incorrecto

- `signo` y `numerador` se crean (e inicializan) independientes para cada invocación
- no trasladan su valor a la siguiente invocación

Modificaremos el diseño para trasladar el valor a la cada iteración

Posibilidades

- añadirlos como atributos
  - no es recomendable, pues son datos internos al método, y no conviene que permanezcan al finalizar éste
- pasarlos como parámetros al método
  - es lo recomendado

Ejemplo correcto

```
método logaritmo (real y, entero i, n,
                  entero signo, real numerador)
retorna real
   {Pre: 0<y<=2, 1<=i<=n, n>0, signo=1 o signo=-1}
      // signo es el valor usado en el término i-1
     // numerador es el valor usado en el término i-1
      var
         real x:=y-1;
         real término;
      fvar
      numerador:=numerador*x;
      signo:=-signo;
      término:=signo*numerador/i;
```


Ejemplo correcto

```
      si i==n entonces
         retorna término;
      si no
         retorna término+
                 logaritmo(y,i+1,n,signo,numerador);
      fsi;

   {Post: x=y-1, valor retornado = } $\sum_{j=i}^{n} (-1)^{j-1} \frac{x^j}{j}$ {
fmétodo

método logaritmo (real y, entero n) retorna real
   logaritmo(y,1,n,-1,1.0);
método
```

Traza del estado al comienzo de cada invocación, y de los valores retornados

y	1.2	1.2	1.2	1.2
i	1	2	3	4
n	4	4	4	4
sig	-1	1	-1	1
num	1.0	0.2	0.04	0.008
term	0.2	-0.02	0.0026	-4.0e-4
v. ret	0.18226	-0.0177	0.00226	-4.0e-4