Bloque 1. Conceptos y técnicas básicas en programación

Notas:

1. Introducción

2. Datos y expresiones. Especificación de algoritmos

3. Estructuras algorítmicas básicas

4. Iteración y recursión

5. Iteración y recursión sobre secuencias

- Descripción de la secuencia. Interfaz. Recorridos sobre secuencias. Búsquedas en secuencias. Esquemas mixtos

6. Iteración y recursión sobre tablas

Introducción

En muchas aplicaciones informáticas es preciso guardar *colecciones* o *listas* de datos

- habitualmente son datos del mismo tipo, o tipos similares
- por ejemplo:
  - lista de alumnos de una asignatura
  - colección de datos de temperatura recogidos por un sensor a lo largo del tiempo
  - vector de 6 dimensiones que representa la posición y orientación de un cuerpo en el espacio

Formas de acceso a una colección

Las dos formas más comunes de acceder a una colección de datos

- ***acceso secuencial*: para acceder a un elemento de la colección es preciso haber accedido a los anteriores**
  - por ejemplo, una cinta de vídeo: para acceder a una imagen es preciso haber accedido a las anteriores
  - acceso a los caracteres introducidos por teclado
  - lectura de los datos de un sensor de temperatura
  - la estructura que admite este acceso se llama ***secuencia***
- ***acceso directo*: se puede acceder a un elemento concreto de la colección directamente, a través de su posición, o índice**
  - por ejemplo, acceso a las imágenes de un DVD
  - acceso a un elemento de un vector
  - la estructura que admite este acceso se llama ***tabla***

Descripción de la secuencia

La *secuencia* representa una colección de cero o más objetos de un determinado tipo, en la que se dispone de acceso secuencial

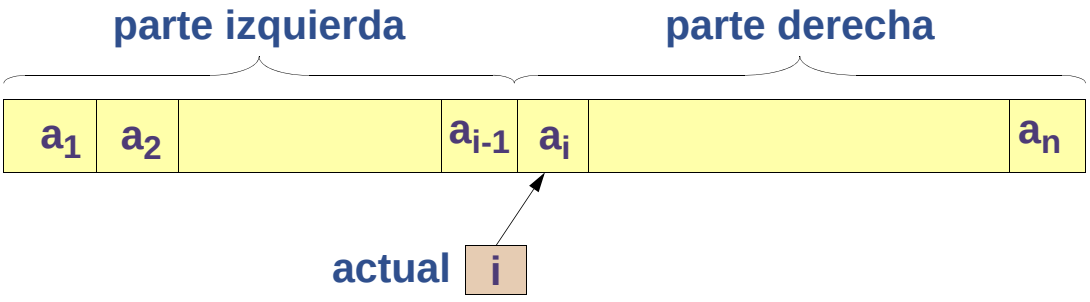
Para el acceso definimos el concepto de elemento *actual*

- es el primero de los elementos que quedan por acceder
  - al terminar de recorrer la secuencia no hay elemento actual

Diferenciamos dos partes en la secuencia

- *parte izquierda*: representa los elementos ya leídos
  - está vacía antes de recorrer la secuencia
- *parte derecha*: representa los elementos aún no leídos
  - el elemento *actual* es el primero de la parte derecha
  - está vacía cuando terminamos de recorrer la secuencia

Descripción de la secuencia (cont.)



Descripción de la secuencia (cont.)

Disponemos de las siguientes operaciones

- ***reinicia***: pone el elemento actual al principio
  - la parte izquierda queda vacía
- ***actual***: retorna el elemento actual
- ***avanza***: el elemento actual pasa a ser el siguiente elemento
- ***fds***: retorna un booleano que indica si estamos en el fin de secuencia

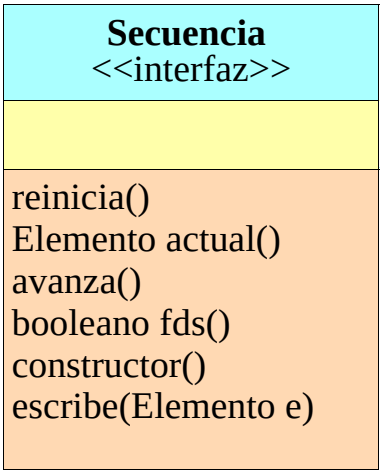
En las secuencias modificables, disponemos además de

- ***constructor***: crea la secuencia vacía
- ***escribe***: añade un elemento a la izquierda del elemento actual
  - lo añade al final si la parte derecha está vacía

Interfaz de la secuencia

La *interfaz* representa una colección de métodos de los que deben disponer las clases que se deriven de ella

- **las *secuencias* deben disponer de los métodos que indican en la interfaz de la figura**
  - **`Elemento` representa el tipo o clase de los elementos que se guardan en la secuencia**



Especificación de los métodos de la secuencia

Dada la secuencia *S<Elemento>* de elementos del tipo *Elemento*:

- *pi(S)*: parte izquierda de *S*
- *pd(S)*: parte derecha de *S*

Especificación de los métodos

- representaremos un elemento de la secuencia con una letra
- representaremos una parte de la secuencia, compuesta por cero o más elementos, con una letra griega

```
método reinicia()
  {Pre:}
  {Post:pi(S)=vacia}
fmétodo
```


Especificación de los métodos de la secuencia

```
método actual() retorna Elemento
  {Pre:pi(S)=α, pd(S)=bβ}
  {Post:pi(S)=α, pd(S)=bβ, valor retornado=b}
fmétodo

método avanza()
  {Pre:pi(S)=α, pd(S)=bβ}
  {Post:pi(S)=αb, pd(S)=β}
fmétodo

método fds() retorna booleano
  {Pre:}
  {Post:valor retornado = pd(S)==vacio)}
fmétodo
```

Especificación de los métodos de la secuencia (cont.)

```
método constructor()
  {Pre:}
  {Post:S está vacía}
fmétodo

método escribe(Elemento e)
  {Pre:pi(S)=α}
  {Post:pi(S)=αe}
fmétodo
```


Problemas a estudiar en las secuencias

1. Recorridos
   - recorrer todos los elementos de la secuencia para hacer algo con cada uno de ellos
2. Búsquedas
   - buscar y obtener un elemento de la secuencia que cumple una determinada condición
3. Combinación de recorridos y búsquedas
   - recorrer una secuencia buscando elementos que cumplen una determinada condición para hacer algo con ellos

Ejemplos de problemas a estudiar en las secuencias

1. Recorridos

- contar el numero de apariciones de una letra en un texto
- calcular la media de una secuencia de enteros
- evaluación de un polinomio representado por la secuencia de sus coeficientes, en un punto *x* dado

2. Búsquedas

- determinar si en un texto aparece una letra dada
- determinar si una secuencia de enteros está formado por números positivos solamente
- mostrar los datos de un alumno cuyo DNI se indica, dada una lista de alumnos

Ejemplos de problemas a estudiar en las secuencias (cont.)

3. Combinación de recorridos y búsquedas

- contar el número de palabras de un texto
- fusionar dos secuencias ordenadas
- sustituir en un texto todas las apariciones de una palabra, por otra

Recorridos sobre secuencias: contar apariciones de una letra en un texto

```
método contarLetra(caracter l,
       Secuencia<caracter> texto) retorna entero
   texto.reinicia();
   var entero num:=0; fvar
   {Inv: num es el número de letras l en pi(texto)}
   {Cota: longitud de pd(texto)}
   mientras no texto.fds() hacer
      si texto.actual()==l entonces
         num++;
      fsi
      texto.avanza();
   fmientras
   retorna num;
{Post: valor retornado=número de letras l del texto}
fmétodo
```


Recorrido sobre secuencias: media de una lista de enteros

```
método media(Secuencia<entero> lista) retorna entero
{Pre: lista tiene al menos un elemento}
   var entero suma:=0; entero num:=0; fvar
   lista.reinicia();
   {Inv: suma es la suma de los enteros de pi(lista),
         num es la longitud de pi(lista)}
   {Cota: longitud de pd(lista)}
   mientras no lista.fds() hacer
      suma:=suma+lista.actual();
      num++;
      lista.avanza();
   fmientras
   retorna suma/num
{Post: valor retornado=media de los enteros de lista}
fmétodo
```

Recorrido sobre secuencias: esquema para el diseño

```
esquema recorrido(Secuencia<Elemento> s)
{Pre:}
   Inicializaciones;
   s.reinicia();
   {Inv: los elementos de pi(s) han sido tratados}
   {Cota: longitud de pd(s)}
   mientras no s.fds() hacer
      tratar s.actual();
      s.avanza();
   fmientras
   Finalización
{Post: los elementos de s han sido tratados}
fesquema
```


Ejemplo: evaluación de un polinomio

Un polinomio P de orden n cuyos coeficientes (de orden mayor a menor) son a_n, a_{n-1}, ..., a_1, a_0 se evalúa para el valor x como

$$P([a_n, \ldots, a_0], x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_i x^i + \ldots + a_1 x + a_0$$

Que se puede expresar también como (regla de Horner)

$$P([a_n, \ldots, a_0], x) = ((((a_n x + a_{n-1})x + a_{n-2})x + \ldots) + a_1)x + a_0$$

Ejemplo: evaluación de un polinomio (cont.)

En definitiva, podemos evaluar los sucesivos polinomios

$$P([a_n], x) = a_n$$

$$P([a_n, a_{n-1}], x) = a_n x + a_{n-1}$$

$$P([a_n, a_{n-1}, a_{n-2}], x) = a_n x^2 + a_{n-1} x + a_{n-2}$$

$$P([a_n, a_{n-1}, a_{n-2}, a_{n-3}], x) = a_n x^3 + a_{n-1} x^2 + a_{n-2} x + a_{n-3}$$

$$\ldots$$

$$P([a_n, \ldots, a_0], x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_i x^i + \ldots + a_1 x + a_0$$

Observamos que cada polinomio es igual al anterior por x, más el siguiente coeficiente

Ejemplo de uso del esquema de recorrido: evaluación de un polinomio

```
método evalua(Secuencia<real> s, real x) retorna real
{Pre: s es una secuencia de coeficientes reales}
   var pol:=0; fvar
   s.reinicia();
   {Inv: pol=P([pi(s)],x)
      (evaluado el polinomio representado por pi(s))}
   {Cota: longitud de pd(s)}
   mientras no s.fds() hacer
      pol:=pol*x+s.actual();
      s.avanza();
   fmientras
   retorna pol;
{Post: valor retornado=P([s],x)}
fmétodo
```

Búsquedas en secuencias: buscar una letra en un texto

```
método tieneLaLetra(caracter l,
                    Secuencia<caracter> texto)
retorna encontrado:booleano
{Pre:}
   encontrado:=false;
   texto.reinicia();
   {Inv: l no está en pi(texto) y
         si encontrado entonces pd(texto)=lα}
   {Cota: longitud de pd(texto)}
   mientras no texto.fds() y no encontrado hacer
      encontrado:= (texto.actual()==l);
      si no encontrado entonces
         texto.avanza();
      fsi
   fmientras
```


Búsquedas en secuencias: buscar una letra en un texto (cont.)

```
   retorna encontrado;

{Post: encontrado= l está en el texto,
       si encontrado entonces l no está en pi(texto) y
            pd(texto)=lα,
        si no encontrado entonces pd(texto)=vacia}
fmétodo
```

Esquema de la búsqueda en secuencia: especificación

```
esquema busca(Secuencia<Elemento> s)
retorna encontrado:booleano
{Pre:}

   Busca un elemento que cumple la propiedad P

{Post: encontrado= algún elemento de s cumple P,
       si encontrado entonces ningún elemento de
            pi(s) cumple P y s.actual() cumple P,
       si no encontrado entonces pd(s)=vacia}
fesquema
```


Esquema de la búsqueda en secuencia: diseño

```
esquema busca(Secuencia<Elemento> s)
retorna encontrado:booleano
   encontrado:=false; Inicializaciones;
   s.reinicia();
   {Inv: ningún elemento de pi(s) cumple P
         si encontrado entonces s.actual() cumple P}
   {Cota: longitud de pd(texto)}
   mientras no s.fds() y no encontrado hacer
      encontrado:= P(s.actual());
      si no encontrado entonces
         s.avanza();
      fsi
   fmientras
   Finalizaciones; retorna encontrado;
fesquema
```

Ejemplo del uso del esquema de búsqueda (1)

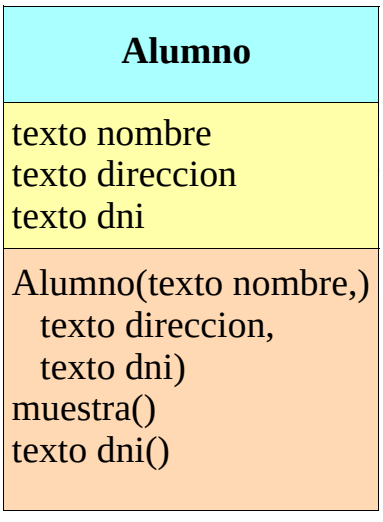
Determinar si una secuencia de números está formada sólo por elementos positivos

```
método todosPositivos(Secuencia<real> s) ret booleano
   var booleano encontrado:=false; fvar
   s.reinicia();
   {Inv: ningún elemento de pi(s) es negativo
        si encontrado entonces s.actual() es negativo}
   mientras no s.fds() y no encontrado hacer
      encontrado:= s.actual()<0;
      si no encontrado entonces
         s.avanza();
      fsi
   fmientras
   retorna no encontrado;
fmétodo
```


Ejemplo del uso del esquema de búsqueda (2)

Mostrar los datos de un alumno cuyo DNI se indica, dada una lista de alumnos

- la clase `Alumno` se caracteriza por el diagrama de clases de la figura
  - al *constructor* se le pasan los valores iniciales de los atributos
  - `muestra` pone los datos del alumno en pantalla
  - `dni` retorna el dni
- Disponemos de una secuencia de objetos de la clase `Alumno`



Ejemplo del uso del esquema de búsqueda (2, cont.)

```
método buscaAlumno(Secuencia<Alumno> s,
                   texto dniBuscado)
   var booleano encontrado:=false; fvar
   s.reinicia();
   {Inv: ningún elemento de pi(s) tiene el dniBuscado
        si encontrado s.actual() tiene el dniBuscado}
   {Cota: longitud de pd(s)}
   mientras no s.fds() y no encontrado hacer
      encontrado:= s.actual().dni==dniBuscado;
      si no encontrado entonces
         s.avanza();
      fsi
   fmientras
```


Ejemplo del uso del esquema de búsqueda (2, cont.)

```
   si encontrado entonces
      s.actual().muestra();
   si no
      muestra mensaje de no encontrado;
   fsi
fmétodo
```

Esquemas mixtos

En los esquemas mixtos combinamos el recorrido con la búsqueda

- buscamos elementos que cumplen una propiedad *P*
- cuando encontramos un elemento que cumple la propiedad, lo tratamos
- después, continuamos el recorrido para buscar más elementos que cumplen la propiedad *P*

Ejemplo de esquema mixto: contar palabras en un texto

El texto contiene letras y separadores

- consideraremos como separadores el espacio en blanco, la coma, y el punto

Especificación

```
método contarPalabras(Secuencia<caracter> texto)
retorna entero
  {Pre: texto contiene letras y separadores}
       contar palabras
  {Post: valor retornado=número de palabras del texto}
fmétodo
```

Ejemplo de esquema mixto: contar palabras en un texto

Diseño

- contaremos una palabra cada vez que pasemos de un separador a una letra
- necesitamos para ello recordar si el carácter anterior era o no un separador: `booleano anteriorEsSep`
- para contar la primera palabra bastará suponer que delante del primer carácter hay un separador
  - inicializar `anteriorEsSep` a `true`

Ejemplo de esquema mixto: contar palabras en un texto (cont.)

```
método contarPalabras(Secuencia<caracter> texto)
retorna entero
  {Pre: texto contiene letras y separadores}
     var
        entero contador:=0;
        booleano anteriorEsSep:=true;
        caracter c;
     fvar
     texto.reinicia();
     {Inv: anteriorEsSep indica si el carácter
           anterior es un separador, contador indica
           el número de palabras de pi(texto)}
     {Cota: longitud de pd(texto)}
```

Ejemplo de esquema mixto: contar palabras en un texto (cont.)

```
      mientras no texto.fds() hacer
         c:=texto.actual();
         si c==' ' o c=='.' o c==',' entonces
            anteriorEsSep:=true;
         si no
            si anteriorEsSep entonces
               contador++;
            fsi
            anteriorEsSep:=false;
         fsi
         texto.avanza();
      fmientras
      retorna contador;
  {Post: valor retornado=número de palabras del texto}
fmétodo
```


Ejemplo de esquema mixto: fusionar secuencias ordenadas

A partir de dos secuencias ordenadas, queremos obtener una tercera secuencia, fusión de las otras dos, y asimismo ordenada

- recorrido de ambas secuencias a la vez
- búsqueda del elemento menor

Especificación

```
método fusionar(Secuencia<E> s1, Secuencia<E> s2)
retorna Secuencia<E>
  {Pre:s1 y s2 son secuencias ordenadas}
      fusionar secuencias
  {Post: valor retornado=secuencia ordenada que
         contiene todos los elementos de s1 y de s2}
fmétodo
```

Ejemplo de esquema mixto: fusionar secuencias ordenadas

```
método fusionar(Secuencia<E> s1, Secuencia<E> s2)
retorna Secuencia<E>
  {Pre:s1 y s2 son secuencias ordenadas}
     var
        Secuencia<E> t= nueva secuencia vacía;
     fvar
     s1.reinicia();
     s2.reinicia();
```


Ejemplo de esquema mixto: fusionar secuencias ordenadas

```
     {Inv: t=fusión(pi(s1),pi(s2))}
     mientras no s1.fds() y no s2.fds() hacer
        // escribir en t el menor de los elementos
        // no leídos de s1 y s2
        si s1.actual()<=s2.actual() entonces
           t.escribe(s1.actual());
           s1.avanza();
        si no
           t.escribe(s2.actual());
           s2.avanza();
        fsi
     fmientras;
     // se cumple Inv y (s1.fds() o s2.fds())
```

Ejemplo de esquema mixto: fusionar secuencias ordenadas

```
      // meter los elementos de s1 que faltan
      mientras no s1.fds() hacer
         T.escribe(s1.actual());
         s1.avanza();
      fmientras;
      // meter los elementos de s2 que faltan
      mientras no s2.fds() hacer
         T.escribe(s2.actual());
         s2.avanza();
      fmientras;
      retorna t;
   {Post: valor retornado=secuencia ordenada que
          contiene todos los elementos de s1 y de s2}
fmétodo
```


Ejercicio propuesto

Sustituir en un texto todas las apariciones de una palabra, por otra