TAD Agenda: Especificación

espec agendas
  usa contactos, booleanos
  género agenda *{Los valores del TAD representan colecciones de contactos a las que se pueden añadir elementos de tipo contacto, y de las que se pueden eliminar sus contactos de uno en uno, eliminándose siempre el último contacto añadido de todos los que contenga la agenda)}*

operaciones

iniciar: → agenda
{Devuelve una agenda vacía, sin contactos}

añadir: agenda a, contacto c → agenda
{Devuelve la agenda resultante de añadir un contacto c a una agenda a.}

vacía: agenda a → booleano
{Devuelve verdad si y sólo si la agenda a está vacía}
...

TAD Agenda: Especificación

borrarUltimo: agenda a → agenda
{Si a no está vacía, devuelve la agenda resultante de eliminar de a el último contacto añadido a ella. Si a está vacía, devuelve la agenda vacía}

está: agenda a, contacto c → booleano
{Dada una agenda a y un contacto c, devuelve verdad si y sólo si en a hay algún contacto igual a c (en el sentido de la operación iguales del TAD contacto), falso en caso contrario}

... {operaciones del iterador}

TAD Agenda: añadiendo operaciones del iterador

. . .

iniciarIterador: agenda a → agenda

{Prepara el iterador para que el siguiente contacto a visitar sea el primero (situación de no haber visitado ningún elemento)}

existeSiguiente?: agenda a → booleano

{Devuelve verdad si queda algún contacto por visitar, devuelve falso si ya se ha visitado el último contacto}

parcial siguiente: agenda a → contacto

{Devuelve el siguiente contacto a visitar.
Parcial: la operación no está definida si no quedan contactos por visitar (not existeSiguiente?(a)}

parcial avanza: agenda a → agenda

{Prepara el iterador para que se pueda visitar el siguiente contacto.
Parcial: la operación no está definida si no quedan contactos por visitar (not existeSiguiente?(a)}

fespec

TAD Agenda: el iterador en C++

```
#ifndef AGENDA_HPP
#define AGENDA_HPP
//Interfaz del TAD. Pre-declaración:
. . .
void iniciarIterador (agenda& a);
bool existeSiguiente (const agenda& a);
bool siguiente (agenda& a, contacto& c);

//Declaración:
struct agenda{
   . . .
   friend void iniciarIterador (agenda& a);
   friend bool existeSiguiente (const agenda& a);
   friend bool siguiente (agenda& a, contacto& c);

  private:
     //campos del struct:
     . . . .


}; . . . .
```

Añadiremos lo necesario a la **representación interna** del tipo, **para** gestionar cuál es el ***estado del iterador***:

- Lo que le añadamos, **únicamente lo usarán las operaciones que implementan el iterador** (y **no** deberán usarlo otras operaciones de la agenda)
- Las otras operaciones de la agenda **no usarán las operaciones del iterador**

Observación

- Especificación:

```
parcial siguiente: agenda a → contacto
{Devuelve el siguiente contacto a visitar.
 Parcial: la operación no está definida si no quedan contactos
 por visitar (not existeSiguiente?(a)}

parcial avanza: agenda a → agenda
{Prepara el iterador para que se pueda visitar el siguiente
 contacto.
 Parcial: la operación no está definida si no quedan contactos
 por visitar (not existeSiguiente?(a)}
```

- Implementación C++: **ambas operaciones juntas**

```
bool siguiente (agenda& a, contacto& c);
{Si existe algún contacto pendiente de visitar, devuelve en
c el siguiente contacto a visitar, y además después avanza el
iterador para que a continuación se pueda visitar otro
contacto, y devuelve true. Si no quedaban contactos
pendientes de visitar, devuelve false.}
```

Ejemplo de uso del iterador:

- Las operaciones del iterador **se usarán fuera de la implementación del TAD**:

```
// Por ejemplo, en el main de P1:
agenda miagenda;
iniciar(miagenda);
//crear contactos y añadirlos a la agenda …
. . . .
//Recorrer todos los contactos de la agenda:
contacto c; bool ok;
iniciarIterador(miagenda);
while (existeSiguiente(miagenda)) {
    ok = siguiente(miagenda,c);
   //tratar el contacto siguiente obtenido en c,
}
. . . .
```

Ejemplo de uso del iterador:

- **Nunca** se debe modificar la colección de datos mientras se recorre con las operaciones de un iterador

```
//… agenda miagenda; contacto c; bool ok; …
//a partir de aquí no se modifica miagenda:
iniciarIterador(miagenda);
while (existeSiguiente(miagenda)) {
     ok = siguiente(miagenda,c);
     //tratar el (siguiente) contacto c ...
    //Durante el recorrido, NO deben añadirse, ni borrar
    //contactos de miagenda
    . . . .
} //fin del recorrido
//ya se puede modificar miagenda
```