```
{Módulo que implementa el TAD "pila genérica".
 Versión: 1.0
 Fecha: 2015-10-07
 Autor: Javier Campos}

módulo genérico pilasGenéricas

parámetro
  tipo elemento

exporta

  tipo pila {Los valores del TAD pila representan secuencias de elementos con
             acceso LIFO (last in, first out), esto es, el último elemento añadido
             será el primero en ser borrado.}

  procedimiento crearVacía(sal p:pila)
  {Devuelve en p la pila vacía, sin elementos}

  procedimiento apilar(e/s p:pila; ent e:elemento)
  {Devuelve en p la pila resultante de añadir e a p}

  función esVacía(p:pila) devuelve booleano
  {Devuelve verdad si y sólo si p no tiene elementos}

  procedimiento cima(ent p:pila; sal e:elemento; sal error:booleano)
  {Si p es no vacía, devuelve en e el último elemento apilado en p y error=falso.
   Si p es vacía, devuelve error=verdad y e queda indefinido}

  procedimiento desapilar(e/s p:pila)
  {Si p es no vacía, devuelve en p la pila resultante de eliminar de p el último
   elemento que fue apilado. Si p es vacía, la deja igual}

  función altura(p:pila) devuelve natural
  {Devuelve el número de elementos de p}

  procedimiento duplicar(sal pilaSal:pila; ent pilaEnt:pila)
  {Devuelve en pilaSal una pila igual a pilaEnt, duplicando la representación
   en memoria}

  función iguales(pila1,pila2:pila) devuelve booleano
  {Devuelve verdad si y sólo si pila1 y pila2 tienen los mismos elementos y en
   las mismas posiciones}

  procedimiento liberar(e/s p:pila)
  {Devuelve en p la pila vacía y además libera la memoria utilizada previamente por p}

  {Las tres operaciones siguientes conforman un iterador interno para la pila}

  procedimiento iniciarIterador(e/s p:pila)
  {Prepara el iterador para que el siguiente elemento a visitar sea un primer
   elemento de p, si existe (situación de no haber visitado ningún elemento)}

  función existeSiguiente(p:pila) devuelve booleano
  {Devuelve falso si ya se han visitado todos los elementos de p; devuelve cierto en
   caso contrario}

  procedimiento siguiente(e/s p:pila; sal e:elemento; sal error:booleano)
  {Si existe algún elemento de p pendiente de visitar, devuelve en e el siguiente
   elemento a visitar y error=falso, y además avanza el iterador para que a
   continuación se pueda visitar otro elemento de p. Si no quedan elementos pendientes
   de visitar devuelve error=verdad y e queda indefinido}
```

implementación

```
  tipos ptDato = ↑unDato;
        unDato = registro
                   dato:elemento;
                   sig:ptDato
                 freg;
        pila = registro
                  cim:ptDato;
                  alt:natural;
                  iter:ptDato {se utiliza para implementar el iterador}
                freg

  procedimiento crearVacía(sal p:pila)
  principio
    p.cim:=nil;
    p.alt:=0
  fin

  procedimiento apilar(e/s p:pila; ent e:elemento)
  variable aux:ptDato
  principio
    aux:=p.cim;
    nuevoDato(p.cim);
    p.cim↑.dato:=e;
    p.cim↑.sig:=aux;
    p.alt:=p.alt+1
  fin

  función esVacía(p:pila) devuelve booleano
  principio
    devuelve p.cim=nil
  fin

  procedimiento cima(ent p:pila; sal e:elemento; sal error:booleano)
  principio
    si esVacía(p) entonces
      error:=verdad
    sino
      error:=falso;
      e:=p.cim↑.dato
    fsi
  fin

  procedimiento desapilar(e/s p:pila)
  variable aux:ptDato
  principio
    si not esVacía(p) entonces
      aux:=p.cim;
      p.cim:=p.cim↑.sig;
      disponer(aux);
      p.alt:=p.alt-1
    fsi
  fin

  función altura(p:pila) devuelve natural
  principio
    devuelve p.alt
  fin
```

```
procedimiento duplicar(sal pilaSal:pila; ent pilaEnt:pila)
variables ptSal,ptEnt:ptDato
principio
  si esVacía(pilaEnt) entonces
    crearVacía(pilaSal);
  sino
    ptEnt:=pilaEnt.cim;
    nuevoDato(pilaSal.cim);
    pilaSal.cim↑.dato:=ptEnt↑.dato;
    ptSal:=pilaSal.cim;
    ptEnt:=ptEnt↑.sig;
    mientrasQue ptEnt≠nil hacer
      nuevoDato(ptSal↑.sig);
      ptSal:=ptSal↑.sig;
      ptSal↑.dato:=ptEnt↑.dato;
      ptEnt:=ptEnt↑.sig
    fmq;
    ptSal↑.sig:=nil;
    pilaSal.alt:=pilaEnt.alt
  fsi
fin

función iguales(pila1,pila2:pila) devuelve booleano
variables pt1,pt2:ptDato; iguales:booleano:=verdad
principio
  si pila1.alt≠pila2.alt entonces
    devuelve falso;
  sino
    pt1:=pila1.cim;
    pt2:=pila2.cim;
    mientrasQue pt1≠nil and iguales hacer
      iguales:=pt1↑.dato=pt2↑.dato;
      pt1:=pt1↑.sig;
      pt2:=pt2↑.sig
    fmq;
    devuelve iguales
  fsi
fin

procedimiento liberar(e/s p:pila)
variable aux:ptDato
principio
  aux:=p.cim;
  mientrasQue aux≠nil hacer
    p.cim:=p.cim↑.sig;
    disponer(aux);
    aux:=p.cim
  fmq;
  p.alt:=0
fin

procedimiento iniciarIterador(e/s p:pila)
principio
  p.iter:=p.cim
fin

función existeSiguiente(p:pila) devuelve booleano
principio
  devuelve p.iter≠nil
fin
```

```
  procedimiento siguiente(e/s p:pila; sal e:elemento; sal error:booleano)
  principio
    si existeSiguiente(p) entonces
      error:=falso;
      e:=p.iter↑.dato;
      p.iter:=p.iter↑.sig
    sino
      error:=verdad
    fsi
  fin

fin
```