Pràctica Estructures de la informació

Codificador - Fase 3

Package genèric II

Monticles i Iterador

Biel Moyà

2013-2014

Monticles

En aquesta tercera fase es demana implementar una cua de prioritats. Aquesta estructura l'implementarem mitjançant un montícle amb les operacions recollides en el fitxer d'especificació *dheap.ads* Cal considerar quina és l'estructura d'un monticle i quina informació és necessària.

Especificació

```
generic
--part generica del paquet
package d_heap is

    type heap is limited private;

    bad_use: exception;
    space_overflow: exception;

    procedure buit      (q: out heap);
    function es_buit    (q: in heap) return boolean;
    procedure posa      (q: in out heap; x: in item);
    procedure elimina_darrer(q: in out heap);
    function darrer     (q: in heap) return item;

    private

    -- definicio
end d_heap;
```

Consideracions

- El montícle ha de permetre emmagatzemar qualsevol estructura, per tant ha de ser genèric.

- En els apunts de classe podeu trobar una implementació molt similar a la que es demana.
- Aquestes operacions són les habituals en la definició de monticles, en el cas de necessitar altres operacions en fases futures, es poden implementar i/o modificar segons les vostres necessitats.
- En aquest punt, seria convenient afegir les excepcions necessàries (veure codi dels apunts).

Iterador

Ja que necessitam respectar la indepèndencia entre la definició d'una estructura i la manera amb que aquesta és implementada, es va decidir que la nostra taula de freqüències era una estructura privada i per tant ens és impossible fer recorreguts sobre la seva informació.

Per posar remei a aquesta situació, ampliarem el paquet de la taula de freqüències creant una nova estructura que ens doni la possibilitat de fer-ho: un iterador.

```
with d_generals;
use d_generals;
package t_frequencies is

   type abecedari is private;

   bad_it: exception;

   procedure tbuida(tfreq: in out abecedari);
   procedure omple (tfreq: in out abecedari;
                    fname: in String;
                    ok: out boolean);
   procedure mostra(tfreq: in abecedari);

   type iterador is private;

   procedure primer(ce: in abecedari; it: out iterador);
   procedure succesor(ce: in abecedari; it: in out iterador);
   procedure consulta(ce: in abecedari; it: in iterador;
                      c: out indexabc; v: out float);
   function  esValid(it: in iterador) return boolean;

   pragma inline(primer,succesor,consulta,esValid);

private
   type abecedari -- la vostra propia estructura ja feta

   -- Definicio iterador


end t_frequencies;
```

Explicació *pragma inline*: La paraula reservada *pragma* ens serveix per donar directives al compilador de com interpretar certes parts del nostre codi.

En aquest cas concret el que pretenem és evitar cridades a funcions molt curtes on el codi màquina per realitzar la cridada és més llarg que el codi de la funció. Per tant el que obtenim amb una funció *inline* és una substitució de la cridada a la funció pel codi de la funció.