Ejercicio 17

`Planta` $= string$
`Stock` $= int$
`Prioridad` $= int$
`Nodo` $=$ `Struct`$\langle$`planta:` $Planta$`, stock:` $Stock$`, prioridad:` $Prioridad\rangle$

```
proc Recolectar (
    in s: Array<Tupla<Planta, Stock>>,
    in u: DiccionarioDigital<Planta, Prioridad>
) : Array<Planta>
```

```
var stockTotal: DiccionarioDigital<Planta, Stock>
    stockTotal:= diccionarioVacio()

// Inicializamos stockTotal
var j:= 0
while (j < s.length) do
  var planta: Planta
  var stock: Stock
  var stockAcumulado: Stock
      planta:= s[j][0]
      stock:= s[j][1]
      stockAcumulado:= 0

  if (stockTotal.pertenece(planta)) then
      stockAcumulado:= stockTotal.obtener(planta)
  endif

  stockTotal.definir(Planta, stockAcumulado + stock)
  j:= j + 1
endwhile

// Evitemos a toda costa iterar sobre un dict
// justificar que se amortiza es siempre complicado
// TRUCAZO
var nodos: ListaEnlazada<Nodo>
    nodos:= listaVacia()
    j:= 0

while (j < s.length) do
  var planta: Planta
      planta:= s[j][0]
  if (stockTotal.pertenece(planta)) then
    var prioridad: Prioridad
    var stock: Stock
        prioridad:= u.obtener(planta)
        stock:= stockTotal.obtener(planta)

    nodos.agregarAtras(new Nodo(planta=planta, stock=stock, prioridad=prioridad))
    stockTotal.borrar(planta)
  endif
  j+= j + 1
endwhile

var arreglo_nodos: Array<Nodo>
    arreglo_nodos:= toArray(nodos)

mergeSort(arreglo_nodos) // ordenar de menor a mayor por stock
mergeSort(arreglo_nodos) // ordenar de mayor a menor por prioridad


var res: Array<Planta>
    res:= new Array(arreglo_nodos.longitud())
    j:=0

while (j < arreglo_nodos.longitud()) do
  res[j]:= arreglo_nodos[j].planta
  j:= j + 1
endwhile

return res
```

Ejercicio 18

`Libreta` $= int$
`Nota` $= int$
`Promedio` $= float$
`Contador` $= totalNotas : int, notasAcumuladas : Nota$

`proc OrdenarPorLibretaYPromedios` (in s: `Array<Tupla<Libreta, Nota>>`) : `Array<Tupla<Libreta, Promedio>>`

```
var promedios: DiccionarioDigital<Libreta, Contador>
promedios:= diccionarioVacio()

var j: int
    j:= 0

// Inicializamos el dict.
while (j < s.length) do
  var libreta: Libreta
      libreta:= s[j][0]
  promedios.definir(
    libreta,
    new Contador(s[j], new Contador(totalNotas=0, notasAcumuladas=0))
  )
  j += 1
endwhile

// Insertamos las notas acumuladas.
j:= 0
while (j < s.length) do
  var libreta: Libreta, nota: Nota
      libreta:= s[j][0], nota:= s[j][1]

  var contador: contador
      contador:= promedios.obtener(libreta)

  contador.totalNotas:= contador.totalNotas + 1
  contador.notasAcumuladas:= nota

  // No hace falta re-definir, hay aliasing
  j += 1
endwhile

// Creamos res
var res: Array<Tupla<Libreta, Promedio>>
    res:= new Array<promedios.tamaño()

// Vamos a zafar de iterar promedios, nunca queremos
// iterar y tener que justificar que las operaciones
// se amortizan, TRUCAZO:
var i: int
    i:= 0
    j:= 0
while (j < s.length) do
  var notaAcumulada: Nota
  var libreta: Libreta
      nota:= s[j][1]
      libreta:= s[j][0]

  if (promedios.pertenece(libreta))
    var contador: contador
        contador:= promedios.obtener(libreta)

    res[i]:= <libreta, contador.notasAcumuladas / contador.totalNotas>
    promedios.borrar(libreta)

    i:= i + 1
  endif

  j:= j + 1
endwhile

mergeSort(res) // ordenar res por la primera coordenada: libreta. (menor a mayor)
mergeSort(res) // ordenar res por la segunda coordenada: promedio. (mayor a menor)

return res
```

Ejercicio 19

`Prioridad` $= int$
`Nodo` $= prioridad : Prioridad, valor : int$

`proc OrdenarSegunCriterio` (in s: `Array<int>`, in crit: `Array<Prioridad>`) : `Array<int>`

```
var s1: ListaEnlazada<Nodo>
var s2: ListaEnlazada<int>
var critDict: DiccionarioLog<int, Prioridad>
   s1:= listaVacia()
   s2:= listaVacia()
   critDict:= diccionarioVacio()

var prioridad: int
   prioridad:= 0

// Inidicalizamos crit dict.
while (prioridad < crit.length) do
 critDict.definir(crit[prioridad], prioridad)
 prioridad:= prioridad + 1
endwhile

var j: int
   j:= 0

// Insertamos en las listas s1 y s2.
while (j < s.length) do
 if critDict.pertenece(s[j]) then
   s1.agregarAtras(s[j], new Nodo(prioridad=critDict.obtener(s[j]), valor=s[j])
 else
   s2.agregarAtras(s[j])
 endif
endwhile

// Ordenar la primera parte del arreglo
var s1Arreglo: Array<Nodo>
var s1Valores: Array<int>
   s1Arreglo:= toArray(s1)
   s1Valores:= new Array(s1Arreglo.length)

mergeSort(s1Arreglo) // ordenar s1Arreglo por prioridad.
// No hay criterio de desempate pues crit no tiene repetidos

j:= 0
while (j < s1Arreglo.length) do
 s1Valores[j]:= s1Arreglo[j].valor
 j:= j + 1
endwhile

// Ordenar la segunda parte del arrego
var s2Valores: Array<int>
   s2Valores:= toArray(s2)

mergeSort(s2Valores) // ordenar s2Arreglo de menor a mayor

// Concatenar en res.
var res: Array<int>
   res:= concat(s1Valores, s2Valores)

// Devolver res.
return res
```