Ejercicio 2.-

Suponiendo conocidas las operaciones = =: carácter carácter-> bool, que comprueba si dos caracteres son iguales, y mayúscula:letra->letra, que devuelve la letra dada en mayúscula, ampliar la especificación del TAD básico PILA[LETRAS] para incluir las siguientes operaciones (pueden ser parciales):
a) cuantas_vocales: pila→natural, que devuelve el total de vocales que contiene la pila.
b) solo_vocales: pila→pila, que elimina de la pila todas las letras que no son vocales.
c) es_palabra: pila->bool, comprueba si las letras de la pila forman una palabra, es decir, no hay dos vocales o dos consonantes seguidas.
Opcional: Escribir los apartados a) y b) utilizando un algoritmo recursivo.

Funciones auxiliares:

Fun es_vocal?(char letra)->bool

```
{Función auxiliar para comprobar que una letra es una vocal}
May<- mayúscula(letra)
Devolver (==('A',may) OR ==('E',may) OR ==('I',may) OR ==('O',may) OR ==('U',may))
Ffun
```

Fun recuperar (pila pilaaux, pila piladestino)->pila

```
{Función auxiliar para pasar el contenido de una pila a otra. Usada para pasar los datos de una pila auxiliar a la original}
Mientras !vacía?(pilaaux) hacer
        e<- cima(pilaaux)
        apilar(e,piladestino)
        Desapilar pilaaux
Fmientras
Devolver piladestino
Ffun
```

a) Fun cuantas_vocales(pila pila)->nat

```
{Función recursiva. No recupera la pila original}
        Si vacía?(pila) entonces devolver 0
        Si no
                e<-cima(pila)
                desapilar(pila)
                Si es_vocal?(e) entonces
                        Devolver 1+cuantas_vocales(pila)
                Si no
                        Devolver cuantas_vocales(pila)
                Fsi
        Fsi
Ffun
```

b) Fun solo_vocales(pila pila)->pila

```
{Función recursiva. No recupera la pila original}
Si vacía?(pila) entonces devolver pilavacía()
Si no
        e<-cima(pila)
        desapilar(pila)
        Si es_vocal?(e) entonces
                Devolver apilar(e,solo_vocales(pila))
        Si no
                Devolver solo_vocales(pila)
        Fsi
Fsi
Ffun
```

c) Fun es_palabra(pila pila)->bool

```
Si vacía?(pila) devolver F
Fsi

paux<-pilavacía()
{añado la primera letra de la pila a la pila aux, y hago que sea la letra anterior}
e<-cima(pila)
desapilar(pila)
añadir(e,paux)
ant<-e
resultado<-T

        Mientras !vacía(pila) y resultado hacer
                e<-cima(pila)
                Si (es_vocal?(e) y es_vocal?(ant)) OR (!es_vocal?(e) y !es_vocal?(ant))
entonces resultado<- F
                {Si no resultado<- T}
                Fsi
                añadir(e,paux)
                ant<-e
                desapilar(pila)
        Fmientras

        recuperar(paux,pila) {recuperar la pila original}
        Devolver resultado
Ffun
```

Ejercicio 4.-

Dar la especificación del TAD COLA[ELEMENTO]. Ampliar dicha especificación creando un tipo nuevo doblecola para la gestión de dos colas a la vez, que permita las operaciones:
a) descomponer: cola->doblecola , que pone los elementos que hay en una cola repartiéndolos de manera alterna en una doblecola;
b) mezclar: doblecola -> cola, que mezcla alternativamente los elementos de una doblecola en una única cola.
Opcional: Escribir uno de los dos apartados utilizando un algoritmo recursivo.

Todas las operaciones con las doblecolas serán iguales que las del TAD cola, pero se añadirá el número de la cola a la que se accede. Esto lo hice ya que en el enunciado se trata a las "doblecolas" como un único objeto, por lo que me pareció mejor que simplemente introducir como parámetros dos colas distintas; en ese caso serían dos colas, no una "doblecola" . Por si fuera necesario, especifiqué el TAD doblecola:

GENERADORAS:
{crear una doblecola vacía} doblecolavacía: -> doblecola
{poner un elemento en la cola} encolar: elemento doblecola número -> doblecola

MODIFICADORAS:
{quitar un elemento de la cola } parcial desencolar: doblecola número -> doblecola

OBSERVADORAS:
{ver el principio de la cola} parcial primero: cola número -> elemento
{ver si la cola está vacía} vacía?: doblecola número -> bool

a) Fun descomponerrec(cola cola, doblecola dc,bool alternar)-> doblecola

```
{Algoritmo recursivo}
Si vacía?(cola) entonces devolver dc
Si no
        e<-primero(cola)
        desencolar(cola)
        Si alternar entonces
                encolardc(e, dc,1)
        Si no
                encolardc(e, dc,2)
```

```
            Fsi
            Devolver descomponerrec(cola,dc, !alternar)
        Fsi
Ffun

Fun descomponer(cola cola)->doblecola {función llamadora de la que contiene el algoritmo recursivo. Hecha para que los parámetros de entrada sean los indicados en el enunciado}
        dc<-doblecolavacia()
        resultado<-descomponerrec(cola,dc,T)
        Devolver resultado
Ffun
```

b) Fun componer(doblecola dc)-> cola {no se recuperan las dc.}

```
        alternar<-T
        cola<-colavacia()

        Si vacía?(dc,1) AND vacía?(dc,2) entonces devolver cola
        Si no
           Mientras !vacía?(dc,1) OR !vacía?(dc,2) hacer
                Si vacía?(dc,1) entonces num<-2
                Si no
                        si vacía?(dc,2) entonces num<-1
                        Si no
                                Si alternar entonces num<-1
                                Si no num<-2
                                Fsi
                        Fsi
                Fsi
                encolar(primero(dc,num),cola)
                desencolar(dc,num)
                alternar<-!alternar
            Fmientras
        Devolver cola
Ffun
```