Tecnología de la programación
Sesión 22

Deberes para la evaluación continua

1. De esta sesión no hay.

Objetivos de la sesión

1. Resolución de ejercicios de TADs Pila, Cola y Lista.

Guion

En esta sesión vamos a realizar 4 ejercicios de las hojas. Todos consisten en **programar como USUARIOS** (es decir, conociendo la especificación, pero no cómo está implementado). Tenéis que intentarlos hacer por vosotros mismos, y luego ver si os coincide la solución. Los ejercicios son:

- Diseñar una acción que invierta el contenido de una pila **(Hoja 6, ejercicio 4)**
- Combinar dos pilas de enteros ordenadas de menor a mayor en una sola pila ordenada de menor a mayor **(Hoja 6, ejercicio 12)**
- Utilizando las operaciones básicas de colas y pilas, escribir un subalgoritmo que obtenga una cola con los elementos de otra cola en orden inverso **(Hoja 6, ejercicio 15)**
- Construir una función que indique si un entero d está en una lista L o no **(Hoja 7, Parte I, ejercicio 6)**

Solución invertir pila

Si nos damos cuenta, el ejercicio no dice nada de si la pila puede quedar destruida. Muestro la solución como si NO pudiésemos destruir la pila original.

```
acción invertirPila (e/s pila p1, e/s pila p2)
{Pre: p1 está iniciada}
{Post: p2 tiene los elementos de p1 en orden inverso, p1 no se destruye}
variables
    pila aux
principio
    iniciarPila(p2)
    iniciarPila(aux)
    mientras que NOT pilaVacia(p1)
        apilar(p2, cima(p1))
        apilar(aux, cima(p1))
        desapilar(p1)
    fmq
    mientras que NOT pilaVacia(aux)
        apilar(p1,cima(aux))
        desapilar(aux)
    fmq
fin
```

Podríais poner también que p2 es de salida únicamente. En general, tendemos a poner siempre e/s en los TADs (cuando trabajamos como usuarios) para evitar errores.

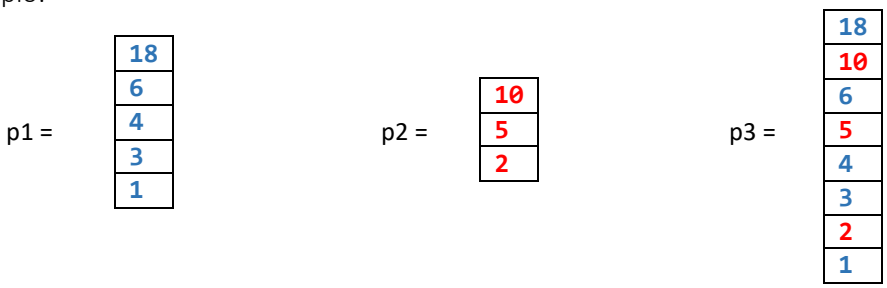
También podríamos haber usado la acción copiarPila. En ese caso, como la operación copiarPila tiene el primer parámetro de entrada (gracias a que pertenece a la especificación) y estamos seguros que no vamos a desapilar nada en p1, entonces podemos poner que p1 es de entrada.

```
acción invertirPila (e/ pila p1, e/s pila p2)
{Pre: p1 está iniciada}
{Post: p2 tiene los elementos de p1 en orden inverso, p1 no se destruye}
variables
    pila aux
principio
    iniciarPila(p2)
    copiarPila(p1, aux) //El método copiarPila ya se encarga de iniciar la pila
    mientras que NOT pilaVacia(aux)
        apilar(p2, cima(aux))
        desapilar(aux)
    fmq
fin
```

Solución combinar pilas ordenadas

El enunciado es un poco ambiguo. ¿Qué significa que una pila está ordenada de menor a mayor? Nosotros, vamos a interpretarlo como que el último elemento apilado es el más grande (se podría hacer al revés).

Ejemplo:




```
acción combinarPilas (e/s pila p1, e/s pila p2, e/s pila p3)
{Pre: p1 y p2 son pilas que están iniciadas y ordenadas de menor a mayor}
{Post: p3 contiene los elementos de p1 y p2 ordenados de menor a mayor}
variables
    pila aux
principio
    mientras que NOT pilaVacia(p1) AND NOT pilaVacia (p2)
        si cima(p1) >= cima (p2)
            apilar(aux, cima(p1))
            desapilar(p1)
        si_no
            apilar(aux, cima(p2))
            desapilar(p2)
        fsi
    fmq
    mientras que NOT pilaVacia(p1)
        apilar (aux, cima(p1))
        desapilar(p1)
    fmq
    mientras que NOT pilaVacia(p2)
        apilar (aux, cima(p2))
```

```
        desapilar(p2)
    fmq
    invertirPila(aux, p3) //El del ejercicio anterior
fin
```

Solución invertir cola

De nuevo, el enunciado no dice nada de si podemos destruir la cola original. En este caso, por variar, vamos a mostrar la solución como que sí la destruimos. Si queremos evitar destruirla, podemos realizar copias antes con el método copiarCola o usar otra cola auxiliar. La clave de este ejercicio es usar una pila auxiliar para invertir fácilmente.

```
acción invertirCola (e/s cola c1, e/s cola c2)
{Pre: c1 está iniciada}
{Post: c2 tiene los elementos de c1 en orden inverso, c1 puede quedar
destruida}
variables
    pila aux
principio
    iniciarCola(c2)
    mientras que NOT colaVacia(c1)
        apilar(aux, primero(c1))
        eliminar(c1)
    fmq
    mientras que NOT pilaVacia(aux)
        añadir(c2, cima(aux))
        desapilar(aux)
    fmq
fin
```

Solución buscar en lista

No nos dicen nada de si la lista está ordenada. Así que lo haremos como si no lo estuviese. En caso de estarlo, podríamos añadir otra condición en el bucle para parar antes.

```
función buscar (lista l, entero d) devuelve booleano
{PRE: l es una lista iniciada}
{POST: devuelve verdad si la lista l contiene al elemento d, falso en caso
contrario, la lista l no se destruye}
variables
    booleano encontrado
    entero i
principio
    encontrado = falso
    i=1
    mientras que i <= longitud(l) AND NOT encontrado
        si extraer(l,i) == d entonces
            encontrado=verdad
        fsi
        i++
    fmq
fin
```