Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2692 - Laboratorio de Algoritmos y Estructuras de Datos II
Trimestre Abril-Julio 2021

Implementaciones del TAD Cola y TAD Pila

1. Introducción

El primer objetivo del laboratorio es el de la implementación de la estructura de datos *lista circular*. El segundo objetivo consiste en realizar implementaciones concretas de las especificaciones del TAD Cola y del TAD Pila. En la representación de la especificación de ambos TADs, se va hacer uso de la estructura **secuencia**.

2. Secuencia y sus operaciones

Una **secuencia** se define como una colección de elementos, en la cual el orden y la multiplicidad son importantes [2]. Para detalles de la definición, características y operaciones de las **secuencias**, ver [2]. La Figura 1 se muestra las operaciones de las **secuencias**. La Figura 2 muestra ejemplos de las operaciones de las **secuencias**.

$\#q$	The number of elements in q.
$e{:}q$	The sequence whose first element is e, and whose subsequent elements are those of q. We have $(e{:}q)[0] = e$ and for $0 < i \leq \#q$, $(e{:}q)[i] = q[i-1]$.
$q1 +\!\!+ q2$	The sequence that begins with $q1$ and carries on with $q2$. We have $(q1 +\!\!+ q2)[i]$ equals $q1[i]$, if $0 \leq i < \#q1$, and equals $q2[i - \#q1]$ if $0 \leq i - \#q1 < \#q2$.
$\mathsf{hd}\ q$	The first element of q, provided q is not empty. We have $\mathsf{hd}(\langle e\rangle +\!\!+ q) = e$.
$\mathsf{tl}\ q$	The second and subsequent elements of q, provided q is not empty. We have $\mathsf{tl}(\langle e\rangle +\!\!+ q) = q$.
$\mathsf{fr}\ q$	All but the last element of q, provided q is not empty. We have $\mathsf{fr}(q +\!\!+ \langle e\rangle) = q$.
$\mathsf{lt}\ q$	The last element of q, provided q is not empty. We have $\mathsf{lt}(q +\!\!+ \langle e\rangle) = e$.

Figura 1: Definiciones de las operaciones de la **secuencia**. Fuente [2]

$$
\begin{array}{rcl}
\#\langle\rangle & = & 0 \\
\langle 1,2\rangle +\!\!+ \langle\rangle & = & \langle 1,2\rangle \\
\langle\rangle +\!\!+ \langle 1,2\rangle & = & \langle 1,2\rangle \\
1{:}\langle 2,3\rangle & = & \langle 1,2,3\rangle \\
\langle 1\rangle +\!\!+ \langle 2,3\rangle & = & \langle 1,2,3\rangle \\
\mathsf{hd}\langle 1,2,3\rangle & = & 1 \\
\mathsf{tl}\langle 1,2,3\rangle & = & \langle 2,3\rangle \\
\mathsf{fr}\langle 1,2,3\rangle & = & \langle 1,2\rangle \\
\mathsf{lt}\langle 1,2,3\rangle & = & 3
\end{array}
$$

Figura 2: Ejemplos de las operaciones de la **secuencia**. Fuente [2]

3. Estructura de datos `Lista`

Se quiere que implemente la estructura de datos **lista doblemente enlazada circular** o **lista circular**. La **lista circular** es una estructura de datos que permite la implementación de la estructura matemática **secuencia**. La Figura3 muestra la representación de la **secuencia** $\langle 9, 16, 4, 1\rangle$ como una **lista circular** con un centinela.

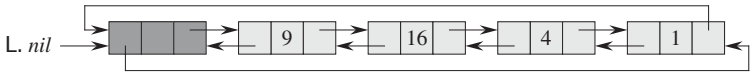

Figura 3: Ejemplo de una lista enlazada, con los elementos 9, 16, 4 y 1. El centinela *L.nil* aparece entre la cabeza y la cola de la lista. Fuente [1]

4. TAD Cola y TAD Pila

La Figura 4 muestra la especificación del TAD Cola, y la Figura 5 muestra la especificación del TAD Pila. Observe que si se realiza una implementación del TAD Cola como una clase, entonces el procedimiento `crearCola` corresponde al constructor de la clase. De forma análoga, el procedimiento `crearPila` es el constructor de una implementación del TAD Pila como una clase.

5. Actividades a realizar

Como puede observar tanto en el TAD Cola de la Figura 4, como en el TAD Pila de la Figura 5, los elementos que son contenidos en ambas, son elementos de tipo T. En nuestras implementaciones, el TAD Cola y el TAD Pila van a contener elementos de tipos enteros.

Debe realizar una implementación concreta del TAD Cola en un archivo llamado `ColaArreglo.kt`. Este archivo contiene la clase `ColaArreglo`, en donde la **secuencia** de elementos de la representación del TAD Cola, va a ser implementada como un arreglo. Debe realizar una segunda implementación del TAD Cola, en un archivo llamado `ColaLista.kt`. Este archivo contiene la clase `ColaLista` en donde la **secuencia** de la representación del TAD Cola, debe ser implementada como una **lista circular**, como la mostrada en la Figura 3. Observe que en la especificación del TAD Cola, en la Figura 4, se tiene que la función `crearCola` genera una nueva instancia del TAD. Ese procedimiento corresponde al constructor de la clase `ColaArreglo` y de la clase `ColaLista`, por lo que no es necesario que sea implementado como un método de esas clases.

También debe hacer dos implementaciones concretas del TAD Pila. Debe realizar la implementación concreta del TAD Pila en un archivo llamado `PilaArreglo.kt`. Este archivo contiene

Especificación del TAD Cola

Modelo de Representación

const MAX : Entero

var $contenido$: Secuencia

Invariante de Representación

$MAX > 0 \land \#contenido \leq MAX$

Operaciones

fun $crearCola$ (**in** n : Entero) → Cola

{ **Pre:** $n > 0$ }

{ **Post:** $crearCola.contenido = \langle\ \rangle \ \land \ crearCola.MAX = n$ }

proc $encolar$ (**in-out** c : Cola, **in** $e : T$)

{ **Pre:** $\#c.contenido < c.MAX$ }

{ **Post:** $c.contenido = c_0.contenido \Vdash \langle e \rangle$ }

proc $desencolar$ (**in-out** c : Cola)

{ **Pre:** $\#c.contenido \neq \langle\rangle$ }

{ **Post:** $c.contenido = \text{tl}\ c_0.contenido$ }

fun $primero$ (**in** c : Cola) → T

{ **Pre:** $\#c.contenido \neq \langle\rangle$ }

{ **Post:** $primero = \text{hd}\ c.contenido$ }

fun $estaVacia$ (**in** c : Cola) → Booleano

{ **Pre:** True }

{ **Post:** $estaVacia \equiv (c.contenido = \langle\rangle)$ }

fun $toString$ (**in** c : Cola) → String

{ **Pre:** True }

{ **Post:** $toString$ = String que contiene una representación de $c.contenido$ }

Fin del TAD Cola

Figura 4: Especificación del TAD Cola

Especificación del TAD Pila

Modelo de Representación

const MAX : Entero

var $contenido$: Secuencia

Invariante de Representación

$MAX > 0 \wedge \#contenido \leq MAX$

Operaciones

fun $crearPila$ (**in** n : Entero) $\rightarrow$ Pila

{ **Pre:** $n > 0$ }

{ **Post:** $crearPila.contenido = \langle\ \rangle \ \wedge \ crearPila.MAX = n$ }

proc $empilar$ (**in-out** p : Pila, **in** $e : T$)

{ **Pre:** $\#p.contenido < p.MAX$ }

{ **Post:** $p.contenido = p_0.contenido \Vdash \langle e \rangle$ }

proc $desempilar$ (**in-out** p : Pila)

{ **Pre:** $\#p.contenido \neq \langle\rangle$ }

{ **Post:** $p.contenido = \text{fr}\ p_0.contenido$ }

fun $tope$ (**in** p : Pila) $\rightarrow T$

{ **Pre:** $\#p.contenido \neq \langle\rangle$ }

{ **Post:** $tope = \text{lt}\ p.contenido$ }

fun $estaVacia$ (**in** p : Pila) $\rightarrow$ Booleano

{ **Pre:** True }

{ **Post:** $estaVacia \equiv (p.contenido = \langle\rangle)$ }

fun $toString$ (**in** p : Pila) $\rightarrow$ String

{ **Pre:** True }

{ **Post:** $toString =$ String que contiene una representación de $p.contenido$ }

Fin del TAD Pila

Figura 5: Especificación del TAD Pila

la clase `PilaArreglo`, en donde la **secuencia** de elementos de la representación del TAD Pila, va a ser implementada como un arreglo. También debe realizar implementación del TAD Pila, en un módulo llamado `PilaLista.kt`. Este módulo contiene la clase `PilaLista` en donde la **secuencia** de la representación del TAD Pila, debe ser implementada como una **lista circular**, como la mostrada en la Figura 3. La función `crearPila` crea una nueva instancia del TAD Pila, por lo tanto ese método corresponde al constructor de la clase por lo que no debe ser implementado como un método aparte.

Debe crear un programa cliente llamado `Main.kt`, que muestre el correcto funcionamiento de las dos implementaciones concretas del TAD Cola y las dos implementaciones concretas del TAD Pila. También debe crear un archivo Makefile que compile todo el código entregado. El cliente debe ser ejecutado con el siguiente comando:

```
>kotlin MainKt
```

Figura 6: Ejecutando el programa cliente de las implementaciones de los TADs Pila y Cola

6. Condiciones de entrega

Los códigos del laboratorio, y la declaración de autenticidad debidamente firmada, deben estar contenidos en un archivo comprimido, con formato *tar.xz*, llamado *LabSem7_X.tar.xz*, donde *X* es el número de carné del estudiante. La entrega del archivo *LabSem7_X.tar.xz*, debe hacerse por medio de la plataforma *Classroom* antes de las 8:00 am del día viernes 02 de julio de 2021.

Referencias

[1] CORMEN, T., LEIRSERSON, C., RIVEST, R., AND STEIN, C. *Introduction to Algorithms*, 3ra ed. McGraw Hill, 2009.

[2] MORGAN, C. *Programming from Specifications*. Prentice Hall, 1998.

Guillermo Palma / gvpalma@usb.ve / Junio de 2021