Pila, Cola, Lista

Objetivos

Primera parte: TAD Lista, Pila, Cola (Especificaciones e implementaciones).

Ejercicio 1 Lista

Considere el siguiente conjunto de operaciones. Las mismas especifican el **TAD Lista** de enteros (que presenta el mismo comportamiento LIFO que una Pila).

```
/* Crea la lista vacía. */
LEnt crearL();

/* Inserta el entero x al principio de la lista. */
void insertar(int x, LEnt & l);

/* Verifica si la lista está vacía. */
bool esVacia(LEnt l);

/* Devuelve el primer elemento de una lista.
Pre:  !esVacia(l) */
int primero(LEnt l);

/* Devuelve la lista l sin su primer elemento.
Pre: !esVacia(l) */
void resto(LEnt & l);
```

(a) Dé una implementación completa (tipo y operaciones) para el **TAD Lista no ordenada** que garantice que el tiempo de ejecución de las operaciones sea $O(1)$.

(b) ¿Qué cambios realizaría si `insFinal` fuera parte del TAD?

```
/* Inserta el elemento x al final de la lista. */
void insFinal( int x, LEnt & l);}
```

(c) ¿Qué cambios realizaría en la **especificación** si se pidiera especificar el **TAD Lista ordenada**?

Ejercicio 2 Cola

(a) Escribir un módulo de especificación para el TAD **Cola (Queue) no acotada** de enteros, conteniendo un conjunto mínimo de operaciones. Desarrolle una especificación haciendo mayormente uso de procedimientos y no de funciones.

(b) Desarrolle una implementación completa de tal manera que el tiempo de ejecución de todas las operaciones sea $O(1)$ en el peor caso.

(c) ¿Qué modificaría en la especificación para el TAD **Cola (Queue) acotada** de enteros? Desarrolle una especificación haciendo mayormente uso de procedimientos y no de funciones.

(d) Desarrolle una implementación completa de **Cola (Queue) acotada** usando una estructura con manejo estático de memoria (la memoria asignada a la estructura no debe variar en tiempo de ejecución) y otra usando una estructura con manejo dinámico de memoria, de tal manera que el tiempo de ejecución de todas las operaciones sea $O(1)$ en el peor caso para ambas implementaciones. Analice las eventuales ventajas y desventajas de ambas implementaciones.

Ejercicio 3 Deque

Una **Deque** es una cola con dos extremos, donde las inserciones y las supresiones pueden efectuarse en cualquiera de ellos.

(a) Escribir un módulo de definición del TAD **Deque no acotada** de enteros que incluya procedimientos para:

I. Crear una Deque vacía.
II. Realizar las inserciones y bajas permitidas.
III. Obtener los elementos situados en los extremos.
IV. Determinar si una Deque es vacía o no.

(b) Desarrolle una implementación completa de tal manera que el tiempo de ejecución de todas las operaciones sea$O(1)$ en el peor caso.

Segunda parte: Aplicaciones.

Ejercicio 4 Ejercicio de examen Diciembre 2018

Se desea saber si una pila y una cola tienen exactamente los mismos elementos y fueron insertados en ellas exactamente en el mismo orden. Para esto implemente la función `mismosElementos` usando las operaciones de los TADs `Pila` y `Cola`. Al terminar, `p` y `c` deben quedar vacías. Los elementos de `Pila` y `Cola` son de un tipo genérico que admite los usuales operadores de comparación.

```
bool mismosElementos(Pila &p, Cola &c);
```

No se deben hacer comparaciones innecesarias. No se pueden usar funciones auxiliares.

Ejercicio 5 Josefo

El libro *The Art of Programming Vol. 1, Donald Knuth* plantea un problema matemático conocido como el problema de Josefo. El mismo lleva el nombre del historiador judío Flavio Josefo y se basa en un relato de lo ocurrido en el sitio de Yodfat donde él y sus camaradas decidieron suicidarse antes de caer en manos de los romanos. Se ubicaron en un círculo, eligieron una posición de inicio y comenzaron a suicidarse en orden (sentido horario), cada 7 personas. Se numeran empeando en 1 y a quien le corresponde el número 7 se suicida. La secuencia vuelve a 1 con el siguiente del que se suicida. El proceso de eliminación de personas del círculo continúa hasta que mueren todos. Según cuenta la leyenda, Josefo (que también era un gran matemático) logró ubicarse en la posición del último en tener que suicidarse, y que en lugar de hacerlo se pasó al lado romano y salvó su vida.

El problema consiste entonces en, dados dos enteros no negativos, n, que representa la cantidad de personas, y k, que indica la cantidad de pasos hasta llegar a la persona que debe suicidarse, determinar cuál es la posición dentro del círculo que ocupará el último sobreviviente.

Para esto es necesario calcular toda la secuencia de ejecuciones.
Por ejemplo, si $n = 8$ y $k = 2$, tendremos un círculo con 8 personas identificadas desde la persona 1 a la persona 8. Las personas que se suicidan según su posición son: 2, 4, 6, 8, 3, 7 y 5, siendo la persona ubicada en la posición 1 la última en suicidarse.

Se pide: implementar una función `Posiciones` que dados un número de personas n y un k devuelva, en $O(n \times k)$, una lista de enteros con la secuencia de posiciones que resuelve el problema de Josefo.

Se sugiere utilizar el TAD Cola de Naturales para representar las posiciones alrededor del círculo.

```
LNat posiciones (unsigned int n, unsigned int k);
/* Retorna la lista de dígitos que representa la secuencia de posiciones que resuelven el problema de Josefo
para n individuos y un k que indica cada cuanto contar, asumiendo que la ejecución comienza para i=1.*/
```

Ejercicio 6 Dígitos (Primer Parcial 2005)

Considere la siguiente representación de números naturales mediante **Listas de Dígitos** en base 10:

- el número natural 1 se representa mediante la lista [1]

- el número natural 15 se representa mediante la lista [1,5]
- el número natural 123 se representa mediante la lista [1,2,3].

Considere la siguiente declaración del tipo `Digito` y las siguientes operaciones sobre Listas de Dígitos:

```
typedef unsigned int Digito;

/* Crea la lista de dígitos vacía */
LDig crear();

/* Inserta un elemento al principio de la lista de dígitos */
LDig cons(Digito x, LDig l);

/* Devuelve true si y solo  si la lista de dígitos esta vacía */
bool esVacia(LDig l);

/* Devuelve el primer elemento de la lista de dígitos. Precondici↩
   ón: la lista no es vacía.*/
Digito primero(LDig l);

/* Devuelve la lista de dígitos sin su primer elemento. ↩
   Precondición: la lista no es vacía.*/
LDig resto(LDig l);
```

Se pide: implementar una función recursiva que calcule la **suma** de dos números naturales representados mediante Listas de Dígitos, con el siguiente cabezal:

`LDig suma(LDig l1, LDig l2);`
/* Devuelve la lista de dígitos que representa al número natural resultado de sumar los dos números naturales parámetros representados mediante listas de dígitos. */

La función `Suma` debe implementarse a partir de las operaciones presentadas arriba y no de definiciones de tipo para `LDig`. Pueden utilizarse funciones auxiliares, siempre y cuando estas sean implementadas recursivamente y, al igual que para Suma, se definan a partir de las operaciones presentadas arriba y no de definiciones de tipo para `LDig`.

A modo de ejemplo, se incluye el resultado de la función para algunos casos:

Entradas	Salida de Suma
[1] [5]	[6]
[1] [9]	[1,0]
[3,5] [1,7]	[5,2]
[2,3] [8,9]	[1,2,2]
[2,3,1] [0,0,0]	[2,3,1]

NOTAS:

- Considere como precondición de su procedimiento que las listas tienen **exactamente** el mismo largo y que estas no pueden ser vacías.
- **No se puede** invertir el orden de los elementos de la lista para resolver el ejercicio

Ejercicio 7 Editor de textos

Usualmente los editores de texto incluyen algún carácter especial (por ejemplo "backspace") que tiene la funcionalidad de borrar el carácter previo no borrado. Por ejemplo, si # es este carácter, entonces el string abc#d##e es en realidad ae. Los editores de texto también incluyen, por lo general, un carácter de borrado de línea. Dicho carácter permite eliminar todos los caracteres previos de la línea de texto. A este carácter lo denotaremos con el símbolo @.

Utilizando el TAD lista de caracteres, escribir un procedimiento que lea y luego imprima una lista de caracteres, que puede contener ocurrencias de los caracteres de borrado arriba descriptos.

Nota: *Deberá definir el TAD lista de caracteres*

Ejercicio 8 Notaciones

Los siguientes son ejemplos de expresiones compuestas de operadores binarios + y *, números naturales y paréntesis, en notación infija y en notación posfija respectivamente:

Expresión infija de entrada: 10 * 5 + 3

Expresión posfija equivalente: 10 5 * 3 +

Expresión infija de entrada: 10 * (5 + 3)

Expresión posfija equivalente: 10 5 3 + *

Expresión infija de entrada: 10 + 5 * 3

Expresión posfija equivalente: 10 5 3 * +

Expresión infija de entrada: (10 + 5) * 3

Expresión posfija equivalente: 10 5 + 3 *

Expresión infija de entrada: (10 + 5 * 10 + 3) * 3

Expresión posfija equivalente: 10 5 10 * 3 + + 3 *

Dada una expresión en notación infija, compuesta de operadores binarios + y *, números naturales y paréntesis, se pide:

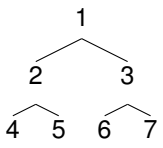
(a) Escribir un procedimiento que transforme la expresión a su representación en notación posfija. Para realizar esta operación use el TAD pila de operadores en donde ir colocando los operadores (según su orden de precedencia). La expresión de entrada es una lista de operadores y operandos representando la expresióninfija. La expresión de salida será una lista de operadores y operandos representando la expresión posfija. Asuma que las expresiones de entrada son sintácticamente correctas.

(b) Evaluar la expresión en notación posfija. Para realizar esta operación utilize el TAD Pila donde ir colocando los operandos. Escribir a tales efectos un procedimiento que reciba la expresión a evaluar (lista de operadores y operandos) y retorne un número natural.

Nota: *Para ambas partes deberá especificar el TAD pila de operandos.*

Ejercicio 9 Niveles

(a) Desarrolle un procedimiento `imprimirPorNiveles` que dado un árbol binario de naturales, representado por una estructura de memoria dinámica, imprima sus elementos por niveles. Esto es, primero deben imprimirse los elementos del primer nivel, luego los del segundo y así sucesivamente, hasta alcanzar la altura del árbol.
Sugerencia: *Utilice el TAD cola de árboles de naturales para resolver el problema.*
Nota: *En caso de necesitar estructuras auxiliares, utilizarlas a través del TAD que corresponda.*
Sea el árbol que se presenta a continuación:



Para este procedimiento se piden dos salidas diferentes:

I. La expresión de salida será una lista de los naturales que se encuentran en los nodos del árbol. Dado el árbol de ejemplo, la salida esperada es la siguiente: `1 2 3 4 5 6 7`

II. La expresión de salida se mostrará por nivel. Es decir, para cada nivel se presentará a su lado la lista de los naturales que se encuentran en los nodos correspondientes a dicho nivel.
Dado el árbol de ejemplo, la salida esperada es la siguiente:

```
1 - 1
2 - 2 3
3 - 4 5 6 7
```

(b) Desarrolle un procedimiento `imprimirPorNiveles2` que dado un árbol binario de naturales, representado por una estructura de memoria dinámica, imprima sus elementos por niveles, pero comenzando desde el nivel más profundo y terminando en el nivel 1, si el árbol es no vacío.

(c) Desarrolle un procedimiento `imprimirPorNiveles3` que dado un **árbol general** de naturales, representado mediante un árbol binario con la semántica *primer hijo siguiente hermano*, imprima sus elementos por niveles, comenzando por el primer nivel, terminando en el nivel más profundo.

Ejercicio 10 Palíndrome con separador

Llamamos *palíndrome con separador* '-' a una secuencia de caracteres de la forma `s-t`, donde '-' es un caracter y `s` y `t` son secuencias de caracteres que **no** contienen el caracter separador ye cumplen que `t` es el reverso de `s`.

Se quiere determinar si una secuencia de caracteres dada, almacenada en una pila de caracteres, es un palíndrome con separador '-'. La longitud de las secuencias de caracteres no está acotada.

Implemente la función

```
bool esPalindromeConSeparador (Pila p)
```

que devuelve `true` si y solo si los caracteres de `p` son un palíndrome con separador '-'.

Ejemplos:

Palíndromes	abac-caba, -, a-a
No palíndromes	abac!caba, bac-caba, abac-cab, -abc, abc-, -a-a-, la secuencia vacía

Solo se pueden usar variables auxiliares de tipo `bool`, `char` o `Pila`.

No se puede usar funciones o procedimientos auxiliares, salvo las operaciones del TAD Pila. Se puede modificar la pila parámetro de entrada `p`. La función `esPalindromeConSeparador` debe ser eficiente, en el sentido de que la ejecución debe terminar si al obtener un elemento de `p` se podría saber que no se cumple la condición pedida.

Ejercicio 11 Máximos relativos

Llamamos *máximos relativos* de una pila aquellos elementos que al ser apilados eran mayores estrictos que todos los que ya estaban en la pila.

Por ejemplo, si en una pila inicialmente vacía se apilan los números 5, 4, 8, 2, 6, 14, 14, y 8, en ese orden, los máximos relativos son el 5, el 8 y el 14 (notar que el primer elemento en ser apilado es un máximo relativo porque no había ninguno mayor o igual a él).

Se debe desarrollar una función que dada una pila parámetro `p` devuelva una pila resultado, `r`, que contenga los máximos relativos de `p`. Los elementos de r deben quedar de tal manera que si se desapilan se obtienen en orden creciente. En el ejemplo anterior de r se desapilarían 5, 8 y 14, en ese orden.

El tiempo de ejecución de la función debe ser $O(n)$, donde n es la cantidad de elementos de `p`.

Se pueden usar variables auxiliares simples, pero no se pueden usar otras estructuras auxiliares (además del parámetro y el resultado), como otra pila, cola o lista.

Al terminar la pila parámetro `p` debe quedar vacía.

(a) Implemente la función

```
Pila maximosRelativos(Pila p)
```

(b) Fundamente por qué se cumple el tiempo de ejecución pedido.