PEC2: Segunda Prueba de Evaluación Continuada

Formato y fecha de entrega

Hay que entregar la solución en un fichero de tipo pdf.

La fecha tope para entregar la solución es el **viernes 4 de abril** (a las 23:59 horas).

Presentación

El propósito de esta segunda PEC es comprobar que has adquirido los conceptos explicados en el capítulo '*Punteros*' de los apuntes '**Estructuras de datos básicas**'.

Competencias

Transversales

- Capacidad de comunicación en lengua extranjera.
- Conocimientos de programación con lenguaje algorítmico.

Específicas

- Capacidad de diseñar y construir algoritmos informáticos mediante técnicas de desarrollo, integración y reutilización.

Objetivos

Los objetivos de esta PEC son:

- Adquirir los conceptos teóricos explicados sobre los punteros.
- Manipular operaciones de los TADs básicos implementados con punteros.
- Diseñar un TAD complejo fruto de la combinación de TADs básicos.
- Diseñar nuevas operaciones en un TAD cualquiera implementado con punteros.

Descripción de la PEC a realizar

Razona y justifica todas las respuestas.

Las respuestas incorrectas **no** disminuyen la nota.

Todos los diseños e implementaciones tienen que realizarse en lenguaje algorítmico. Los nombres de los tipos, de los atributos y de las operaciones se tienen que escribir en inglés. Los comentarios y mensajes de error no es

obligatorio hacerlos en inglés, a pesar de que se valorará positivamente que se haga, puesto que es el estándar.

Recursos

Para realizar esta prueba dispones de los siguientes recursos:

Básicos

- **Apuntes *"Estructuras de datos básicas"*.** En la sección *Recursos* del aula de teoría puedes encontrar un enlace a este material.
- **Foro del aula de teoría.** Dispones de un espacio asociado a la asignatura donde puedes plantear tus dudas sobre el enunciado.

Complementarios

- **Buscador web.** La forma más rápida de obtener información ampliada y extra sobre cualquier aspecto de la asignatura es mediante un buscador web.

Criterios de valoración

Para la valoración de los ejercicios se tendrá en cuenta:

- La adecuación de la respuesta a la pregunta formulada.
- Utilización correcta del lenguaje algorítmico.
- Claridad de la respuesta.
- Completitud y nivel de detalle de la respuesta aportada.

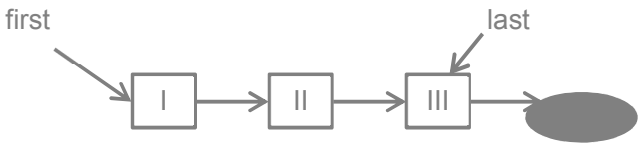
Ejercicio 1: Conceptos básicos (15%)

Tarea: Responde las preguntas siguientes justificando las respuestas.

i) Que ventaja nos ofrece trabajar con memoria dinámica? Qué limitación tiene esta forma de trabajar con la memoria?

ii) Dada una cola implementada con punteros, dibuja la evolución de la estructura siguiente con la operación propuesta:

{ c = < I, II, III }
enqueue(c, IV);



iii) Dada una pila *p* implementada con punteros, dibuja el resultado de aplicar la operación siguiente a la pila p con el estado inicial:

{ p = [}
top (p);

Ejercicio 2: Modificación de TAD básicos (20%)

Tarea: Dada la implementación del TAD lista con punteros (explicada en los apuntes):

```
tipo
        node = tupla
                        e : elem;
                        next : puntero a node;
        ftupla

        list =    tupla
                        first, prev : puntero a node;
        ftupla
ftipo
```

Extiende el tipo añadiendo las operaciones siguientes:

i) **accion** removeLast (entsal l : **list**)

Retorna la lista *l* con el último elemento eliminado. Si la lista está vacía se emite *error*. Para recorrer la lista, utilizad el puntero *l.prev*

ii) **accion** assignPosN (entsal l: **list**, ent p: **entero**, ent e: **elem**)

Realiza una asignación del elemento *e* en la posición *p* de la lista *l*. En otras palabras, substituye el elemento en la posición *p* de la lista *l* por el elemento *e*.

Por ejemplo, el resultado de aplicar la operación *assignPosN* del elemento *x* en la posición 2 en una lista "< a b c d >" será "< a x c d >". Si la lista no tiene *p* elementos como mínimo se emite error. Se asume que *p* contiene un valor mayor que 0.

Para diseñar estas operaciones no has de utilizar las operaciones del tipo lista (create, insert, delete...), sino que has de trabajar directamente con la implementación del tipo que os hemos facilitado en el enunciado.

Ejercicio 3: Transparencia del TAD (20%)

Tarea: Modifica el código de les operaciones teniendo en cuenta los cambios explicados en el capítulo para garantizar la transparencia del tipo.

Rellena los espacios de las siguientes operaciones implementadas en el ejercicio 4 de la PEC1, siguiendo las explicaciones de los apuntes para garantizar la transparencia del tipo independientemente de su implementación (por ejemplo, parámetros de entsal, operaciones destroy, duplicate y equal, ...)

i)

accion selecEditorial (**entsal** b : tBook; **sal** codeEditorial : **entero**)

faccion

ii)

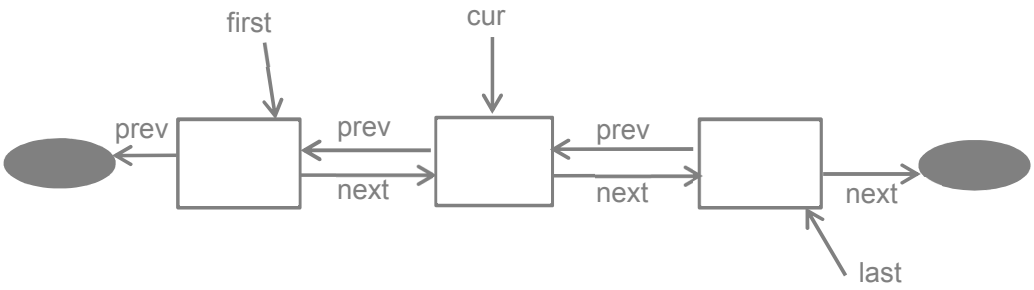
```
accion correctInconsistency (entsal st: tStore)
  var
        c : tCatalogEntry;
        b : tBook;
        found : booleano;
  fvar
        [redacted]
        mientras  ¬ end(st.titles) hacer
              [redacted]
              [redacted]
              found := falso;
              mientras  ¬ end(st.catalog) Λ  ¬ found hacer
                    [redacted]
                    si c.codeBook = b.code entonces
                          found := verdadero;
                    sino
                          [redacted]
                    fsi
              fmientras

              si  ¬ found entonces
                    [redacted]
              sino
                    [redacted]
              fsi
                    [redacted]
        fmientras
faccion
```

Ejercicio 4: Diseño de un tipo con punteros (25%)

Tarea: Hasta ahora hemos trabajado con los tipos de datos abstractos pila, cola y lista, pero a veces estos tipos no nos permiten reflejar la realidad y necesitamos crear tipos más complejos (como por ejemplo, una lista doblemente encadenada).

Definimos el TAD **tDoubleLinkedList** que contiene una lista doblemente encadenada de elementos (**e**). Para facilitar su comprensión, os proporcionamos un ejemplo de como podría ser un caso concreto de este tipo implementado con punteros:



i) Completa la definición de este TAD utilizando punteros:

tipo

tNode = **tupla**

ftupla

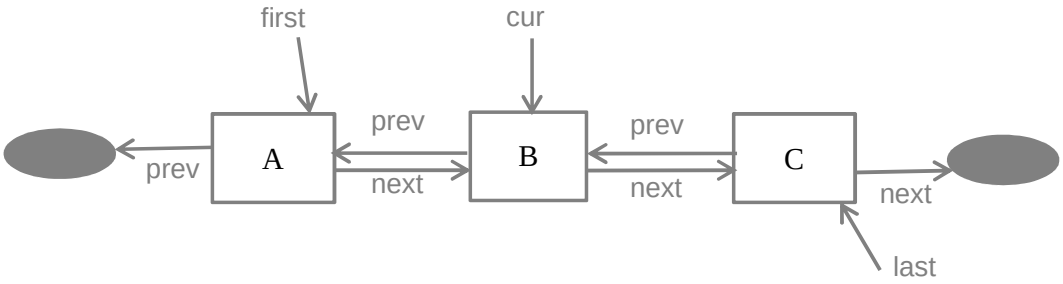
tDoubleLinkedList =

ftipo

Este nuevo TAD ofrece diversas operaciones, entre ellas destacamos la posibilidad de *recorrer la lista en los dos sentidos* (hacia adelante y hacia atrás). Esto nos permitirá implementar la operación de *rotación de elementos.*

La rotación de elementos N lugares a la izquierda consiste en mover todos los elementos N posiciones a la izquierda dentro de la secuencia de forma circular. Si un elemento llega al principio de la secuencia, deberá continuar por el final hasta completar las N posiciones.

ii) Antes de diseñar una operación, siempre es un buen recurso imaginarse como se comportaría la operación en un caso concreto. *Así pues, a partir del esquema facilitado en el enunciado, dibuja el resultado de hacer la siguiente operación: rotateLeftt (dll, 1).*



iii) Implementa la operación a partir de la siguiente cabecera:

accion rotateLeft (**entsal** dll : tDoubleLinkedList; **ent** n : **entero**)

Nota: Si el número n es cero o la lista está vacía o contiene sólo un elemento se retorna la lista *dll* sin cambios.

Ejercicio 5: Diseño de operaciones (20%)

Tarea: El personal de la librería necesita información de las editoriales cuando gestiona los encargos. Concretamente, desea conocer los datos de contacto para hacer el pedido.

Para evitar realizar búsquedas continuas a la estructura que almacena las editoriales a partir del código de la editorial que aparece en las líneas de los encargos **(tOrderLine**), modificaremos los tipos definidos en la solución de la PEC1 con ***la ayuda de los punteros para no replicar los datos almacenados.***

i) Añade un campo nuevo (**infoEditorial**) con los datos de contacto de la editorial:

```
tCatalogEntry = tupla
                    codeBook: entero;
                    numPages: entero;
                    cost: real;
                    yearLastEdition: entero;
                    score: entero;
                    availability: entero;
                    [redacted]
ftupla
```

ii) Substituye los campos **codeBook** y **codeEditorial** de la línea de encargo (**tOrderLine**) por un campo que permita acceder a los datos del catálogo:

```
tOrderLine = tupla
                    bookEntry : [redacted]
                    numItems: entero;
ftupla
```

iii) Implementa la función que retorna la dirección postal de la editorial (**tString**) del libro de una línea de un encargo (**tOrderLine**).

iv) ¿Qué ventaja nos aporta el uso de punteros en este ejercicio? ¿Y qué problema puede aparecer si no vamos con cuidado? Razona las respuestas

Nota: **Propiedad intelectual**

A menudo es inevitable, al producir una obra multimedia, hacer uso de recursos creados por terceras personas. Es por lo tanto comprensible hacerlo en el marco de una PEC de los estudios del Grado, siempre que esto se documente claramente y no suponga plagio en la PEC.

Por lo tanto, al presentar una PEC que haga uso de recursos ajenos, se tiene que presentar junto con ella un documento en que se detallen todos ellos, especificando el nombre de cada recurso, su autor, el lugar donde se obtuvo y su estatus legal: si la obra está protegida por el copyright o se acoge a alguna otra licencia de uso (Creative Commons, licencia GNU, GPL ...). El estudiante tendrá que asegurarse que la licencia que sea no impide específicamente su uso en el marco de la PEC. En caso de no encontrar la información correspondiendo tendrá que asumir que la obra está protegida por el copyright.

Además, se habrán de adjuntar los ficheros originales cuando las obras utilizadas sean digitales, y su código fuente si corresponde.

Otro punto a considerar es que cualquier PEC que haga uso de recursos protegidos por el copyright no podrá en ningún caso publicarse en Mosaic, la revista del Graduado en Multimedia de la UOC, a no ser que los propietarios de los derechos intelectuales den su autorización explícita.