Última entrega de prácticas, correspondiente a la sesión "Tipos Abstractos de Datos"

Enunciado del ejercicio a entregar

Esta entrega consiste en la realización del ejercicio que se enuncia a continuación. Se abrirá una tarea en Studium para que, una vez finalizada la realización del ejercicio, el alumno suba un único fichero `.zip` o `.tgz` que contenga todos los ficheros con el código fuente que se solicitan. Esta tarea en Studium permanecerá abierta hasta el viernes, **3 de diciembre de 2021, a las 23:59**. No obstante, se permitirán entregas retrasadas hasta el lunes, **20 de diciembre, a las 23:59**. Después de esa hora ya no se admitirá ninguna entrega.
A continuación se describe la tarea a implementar.

Enunciado 1

TAD PILA

Tipo especial de lista con la restricción de que las inserciones y eliminaciones solo pueden realizarse en una posición, denominada tope o cima de la lista. Se trata de una lista de tipo LIFO (*Last In, First Out*), en la que existe un elemento (siempre que no este vacía) en la cima de la pila que es el único visible o accesible. A continuación se definen las operaciones básicas que típicamente incluye un TAD pila:

pilaCreaVacía(p): inicia o crea la pila `p` como una pila vacía, sin ningún elemento

pilaVacía(p): devuelve verdadero si la pila `p` está vacía, y falso en caso contrario

pilaInserta(x,p): añade el elemento `x` a la pila `p` convirtiéndolo en el nuevo tope o cima de la pila

pilaSuprime(p): suprime el elemento del tope o cima de la pila y finaliza la función devolviendo el valor suprimido

Partiendo de la especificación anterior, realizar dos implementaciones del TAD Pila utilizando, en la primera, memoria estática y contigua (matrices), y en la segunda memoria dinámica y dispersa (listas simplemente enlazadas), tal y como se detalla a continuación:

►1. Memoria estática (matriz), según muestra la Figura 1.

Como se puede apreciar en Figura 1, la variable `variablePila` es una estructura que contiene dos campos:

- una matriz, cuyas celdas serán de tipo `tipoElemento`
- un entero que señala siempre al último elemento añadido a la pila, es decir, el tope o cima de la pila

La implementación debe hacerse ajustándose a los tipos y prototipos que se adjuntan en el fichero cabecera `pila.h` que se encuentra en la carpeta `matrices` que se entrega junto con este enunciado.

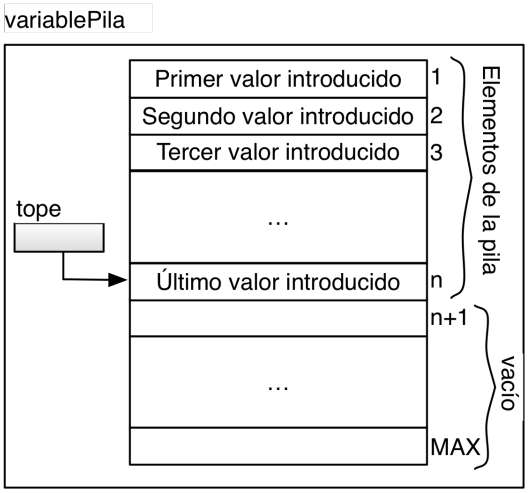

Figura 1: Pila implementada mediante matrices.

►**2.** Memoria dinámica (listas simplemente enlazadas **sin** nodo ficticio), según muestra la Figura 2

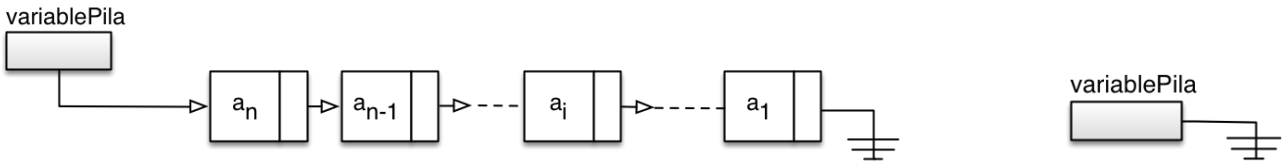

Figura 2: Pila implementada mediante lista símplemente enlazada sin nodo ficticio.

Como se puede apreciar en Figura 2, la variable `variablePila` es un puntero de tipo `tipoCelda` que apunta al nodo que contiene el último elemento (tope o cima) insertado en la pila. Una pila vacía es dicha variable a `NULL` (ver Figura 2).

La implementación debe hacerse ajustándose a los tipos y prototipos que se adjuntan en el fichero cabecera `pila.h` que se encuentra en la carpeta `listasEnlazadas` que se entrega junto con este enunciado.

Enunciado 2

TAD COLA

Tipo especial de lista con la restricción de que las inserciones se realizan por un extremo de la lista denominado fondo y las eliminaciones por el otro extremo, denominado frente. Se trata de una lista de tipo FIFO (*First In, First Out*), en las que el primer elemento que entra es el primero en salir. A continuación se definen las operaciones básicas que típicamente incluye un TAD cola:

colaCreaVacía(c): inicia o crea la cola `c` como una cola vacía, sin ningún elemento.

colaVacía(c): devuelve verdadero si la cola `c` está vacía, y falso en caso contrario

colaInserta(x,c): añade el elemento x a la cola c convirtiéndolo en el último elemento de la cola. Se añade por el fondo o final de la cola.

colaSuprime(c): suprime el primer elemento de la cola, de forma que el siguiente elemento pasa a ser el nuevo frente, y finaliza la función el valor suprimido

Partiendo de la especificación anterior, realizar dos implementaciones del TAD Cola utilizando, en la primera, memoria estática y contigua (matrices), y en la segunda memoria dinámica y dispersa (listas simplemente enlazadas), tal y como se detalla a continuación:

►**1.** Memoria estática (matriz circular), según muestra la Figura 3.

Como se puede apreciar en Figura 3, la variable `variableCola` es una estructura que contiene cuatro campos:

- una matriz, cuyas celdas serán de tipo `tipoElemento`
- un entero que señala siempre al primer elemento de la cola: frente
- un entero que señala siempre al último elemento de la cola: fondo

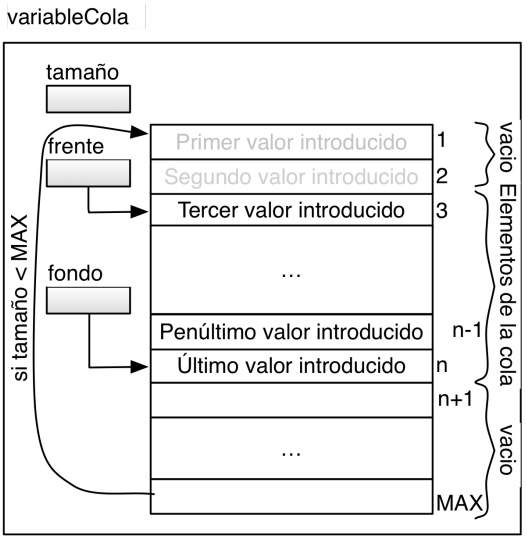

Figura 3: Cola implementada mediante matriz circular.

- un entero que representa el número de elementos existentes en la cola en cada instante, tamaño, y que, tal y como se explica en teoría, se utiliza para dotar de ese carácter circular a la matriz, junto con la implementación de la función `incrementarIndice(int *indice)`. Es importante recordar que se debe implementar esta función, ya que, dado que no pertenece a la interfaz del tipo abstracto, su prototipo no se encontrará en el fichero de cabecera, pero sí es una función que se utiliza dentro de las funciones a implementar.

La implementación debe hacerse ajustándose a los tipos y prototipos que se adjuntan en el fichero cabecera `cola.h` que se encuentra en la carpeta `matrices` que se entrega junto con este enunciado.

▶2. Memoria dinámica (listas simplemente enlazadas **sin** nodo ficticio), según muestra la Figura 4

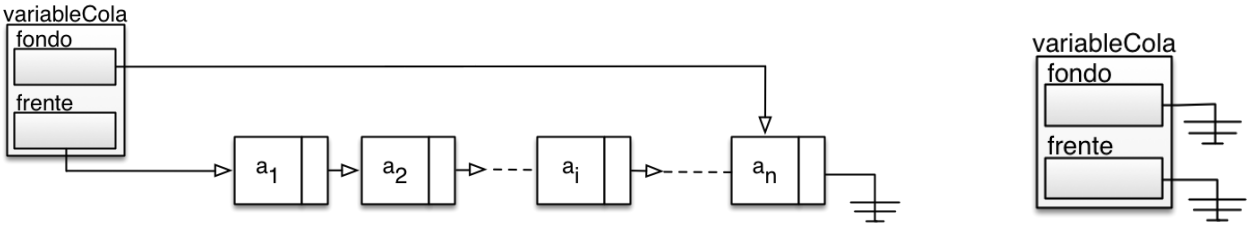

Figura 4: Cola implementada mediante lista símplemente enlazada sin nodo ficticio.

Como se puede apreciar en Figura 4, la variable `variableCola` es una estructura con dos punteros de tipo `tipoCelda`:

- el puntero `frente` que apunta siempre al primer nodo de la lista enlazada, por donde se extraen los elementos.
- el puntero `fondo` que apunta siempre al último nodo de la lista enlazada, por donde se añaden los nuevos elementos.

En el caso de estar vacía la cola, ambos punteros estarán a `NULL` (ver Figura 4).

La implementación debe hacerse ajustándose a los tipos y prototipos que se adjuntan en el fichero cabecera `cola.h` que se encuentra en la carpeta `listasEnlazadas` que se entrega junto con este enunciado.

Consideraciones generales sobre la implementación

Como se ha estudiado en teoría la especificación de un tipo abstracto de datos es única, pero una vez definido puede implementarse de varias formas. Sólo es necesario respetar la especificación para que, fuera de la sección donde se define, pueda utilizarse como si se tratara de un tipo de datos primitivo (encapsulación). La implementación de la práctica se propone de tal manera que, sin hacer cambios en el algoritmo que utiliza los tipos, se puede utilizar una u otra implementación de ambos, poniendo, claramente, de manifiesto el nivel de abstracción conseguido.

En este sentido, y para poder realizar las pruebas de funcionamiento correcto de la práctica, se proporciona el fichero `palindromo.c` que contiene el código necesario para realizar las pruebas, excepto la función `palindromo()` que debe implementar el estudiante. Dicha función, `palindromo()`, debe devolver un 1, si la cadena recibida es un palíndromo, y un 0 si no lo es. La función determinará si una frase es palíndromo, es decir, si se puede leer lo mismo de izquierda a derecha y de derecha a izquierda, una vez eliminados los blancos (Ej.: "anula la luz azul a la luna"). Utilizar el TAD Pila y el TAD Cola, añadiendo cada carácter de la frase a las dos estructuras a la vez. La extracción simultánea de caracteres de ambas y su comparación determinará si la frase es palíndromo o no.

Se debe comprobar que al intercambiar las implementaciones de la pila y la cola (listas enlazadas por matrices y viceversa) el programa sigue funcionando correctamente y produciendo los mismos resultados. Los prototipos de las funciones disponibles en los ficheros de cabecera se han mantenido de manera que el cambio de implementación no implique cambios en las invocaciones de las funciones.

La gestión de errores es responsabilidad del estudiante. Los prototipos de las funciones están preparados para devolver un valor entero que indique un código de error, o bien un 0 en caso de éxito. **Importante**: Las funciones

a implementar no mostrarán ningún mensaje por pantalla, en ningún caso.

El fichero `entregaTAD.zip` que se entrega contiene la siguiente estructura de carpetas:

- `entregaTAD.pdf`, el presente documento de enunciado en formato PDF
- `palindromo.c`, fichero de código C con una función `main()` y todo lo necesario para poder hacer la prueba de funcionamiento de las funciones implementadas. La función que utilizará los TAD implementados, `palindromo()`, también **será escrita** por el estudiante, según lo explicado anteriormente.
- `Makefile`, que permite la compilación y generación de dos ejecutables, `palindromoM`, que utilizará la implementación de los dos TAD mediante matrices, y `palindromoL`, que utilizará la implementación de los dos TAD mediante listas simplemente enlazadas. Está escrito de tal manera que no hay que hacer nada especial, solo ejecutar una de las tres reglas: `make all`, que crea ambos ejecutables, `make palindromoM` que solo creará el ejecutable `palindromoM` y `make palindromoL`, que solo creará el ejecutable `palindromoL`. Hay una última regla, `make clean`, que elimina los dos ejecutables y todos los fichero objeto producidos.
- `matrices`, carpeta que contiene los ficheros de cabecera `pila.h` y `cola.h` con las definiciones de tipos y prototipos de funciones para las respectivas implementaciones con matrices. El estudiante debe escribir, en los ficheros `pila.c` y `cola.c`, el código correspondiente a las operaciones especificadas para cada TAD, utilizando la representación mediante matrices.
- `listasEnlazadas`, carpeta que contiene los ficheros de cabecera `pila.h` y `cola.h` con las definiciones de tipos y prototipos de funciones para las respectivas implementaciones con listas enlazadas. El estudiante debe escribir, en los ficheros `pila.c` y `cola.c`, el código correspondiente a las operaciones especificadas para cada TAD, utilizando la representación mediante listas enlazadas.

Una vez implementadas todas las funciones y probadas como ya se ha indicado anteriormente, se debe crear un nuevo fichero `entregaTAD.zip` que, manteniendo la misma estructura descrita contenga los ficheros con las implementaciones de las funciones en sus respectivas carpetas (`matrices` y `listasEnlazadas`) y el fichero `palindromo.c` con la función implementada para pruebas. Este es el fichero que se debe subir a Studium dentro del plazo establecido: 23:59 horas del viernes, 3 de diciembre de 2021, o bien 23:59 horas del lunes, 20 de diciembre, entrega retrasada.