Tipos de datos lineales.

ED Grupo A. Profesor: Isabel Pita.

Nombre del alumno:

1. (1 punto) Indica lo que muestra por pantalla el siguiente fragmento de código:

```
std::stack<int> p;  stsd::queue<int> q;
p.push(5);
p.push(8);
p.push(10);
std::cout << p.top() << '\n';
q.push(p.top()); p.pop();
q.push(p.top()); p.pop();
std::cout << q.front() << '\n';
p.push(q.front()); q.pop();
p.push(q.front()); q.pop();
std::cout << p.top() << '\n';
std::cout << q.front() << '\n';
```

   Respuesta:

   El código escribe los valores 10, 10, 8 uno en cada línea y luego se produce un error de cola vacía.

2. (1 punto) ¿Se puede conseguir coste constante en todas las operaciones de la pila (`push`, `pop`, `top`, `empty`) si se utiliza una lista enlazada simple, con puntero al comienzo y al final de la lista y se sitúa la cima de la pila al final de la lista?. Justifica tu respuesta.

   Respuesta:

   No. Si la cima de la pila está al final de la lista enlazada, no se puede realizar la operación `pop` en tiempo constante, ya que no se puede colocar el puntero al final de la lista una vez que se elimina el último elemento en tiempo constante. La única forma que tenemos de colocar el puntero en el nuevo final de la lista enlazada (una vez eliminado el elemento) es recorrer la lista desde el principio, lo que tiene coste lineal en el número de elementos.

3. (1 punto) Queremos implementar una cola y para ello sólo contamos con el TAD pila (no existen vectores ni listas enlazadas) ¿Cuál es el menor número de pilas necesario para implementarla?. Indica como se realizaría la función `pop` de la cola implementada de esta manera.

   Respuesta:

   Se necesitan 2 pilas.

   Para realizar la operación `pop`: (llamaremos primera pila a aquella en que se apilan los elementos en la operación `push`.)

   - Se comprueba si las dos pilas son vacías. En este caso se da error de cola vacía
   - Sino, si la segunda pila no es vacía se elimina su cima
   - Sino, si la segunda pila es vacía, entonces la primera no lo puede ser. Se vuelvan todos los valores de la primera pila en la segunda y se elimina el valor que quede en la cima.

4. (1 punto) En la implementación de las colas dobles por medio de una lista enlazada doble, con nodo fantasma y circular, en la cual el puntero `fantasma` apunta al nodo fantasma de la lista enlazada doble, indica cual de las siguientes condiciones denota que la cola doble está vacía:.

   a) `fantasma == nullptr`

   b) `fantasma->sig == nullptr &&  fantasma->ant == nullptr`

   c) `fantasma->sig == fantasma && fantasma->ant == fantasma`

   d) `fantasma->elem == 0`

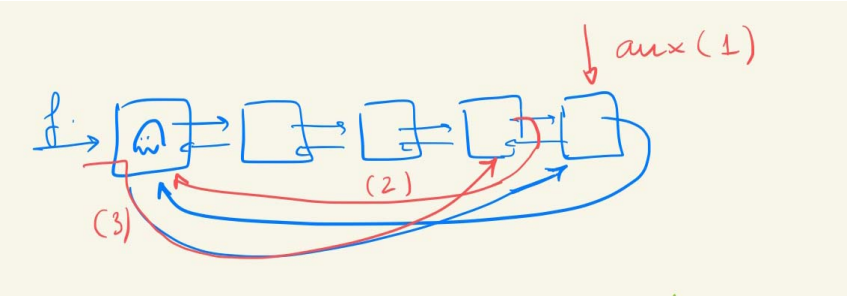
   Respuesta:

   Opción c). La lista doble circular con nodo fantasma es vacía si el puntero anterior y siguiente apuntan al nodo fantasma.

5. (1 punto) Completa el código de la siguiente función que dada una lista enlazada doble con nodo fantasma y circular elimina el último elemento de la lista. El parámetro `f` apunta al nodo fantasma de la lista.

   Respuesta:

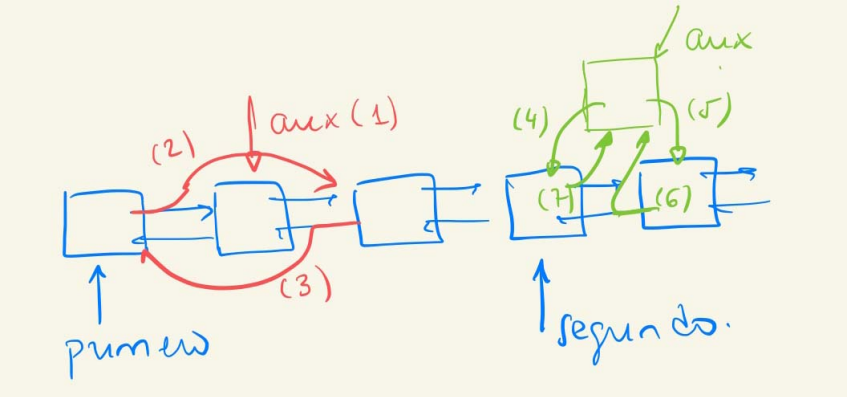
```
void elimina_ultimo (Nodo* f){
       Nodo * aux = f->ant; // Guarda el puntero
        aux->ant->sig = f;  // enlaza la lista que queda
        f->ant = aux->ant;
        delete aux;
}
```



6. (1 punto) Completa el código de la siguiente función que dada una lista enlazada doble, con nodo fantasma y circular, traslada el nodo siguiente al que apunta el primer parámetro detrás del nodo al que apunta el segundo parámetro. Se garantiza que los parámetros apuntan a nodos de la lista.

   Respuesta:

```
Nodo* trasladar (Nodo* primero, Nodo* segundo){
        // desenganchamos el nodo siguiente a primero
        Nodo* aux = primero->sig;
        aux->ant->sig = aux->sig;
        aux->sig->ant = primero;
        // Lo anyadimos detras de segundo
        aux->ant = segundo;
        aux->sig = segundo->sig;
        segundo->sig->ant = aux;
        segundo->sig = aux;
        return primero;
}
```



7. (1 punto) Con la implementación vista en clase de las colas dobles, con una lista enlazada doble, nodo fantasma y circular se consigue coste constante en todas las operaciones de la cola doble: `push_font`, `push_back`, `pop_front`, `pop_back`, `front`, `back`, y `empty`. ¿Se puede conseguir coste constante con una lista enlazada doble con un puntero al principio y al final de la lista?. Justifica tu respuesta.

   Respuesta:

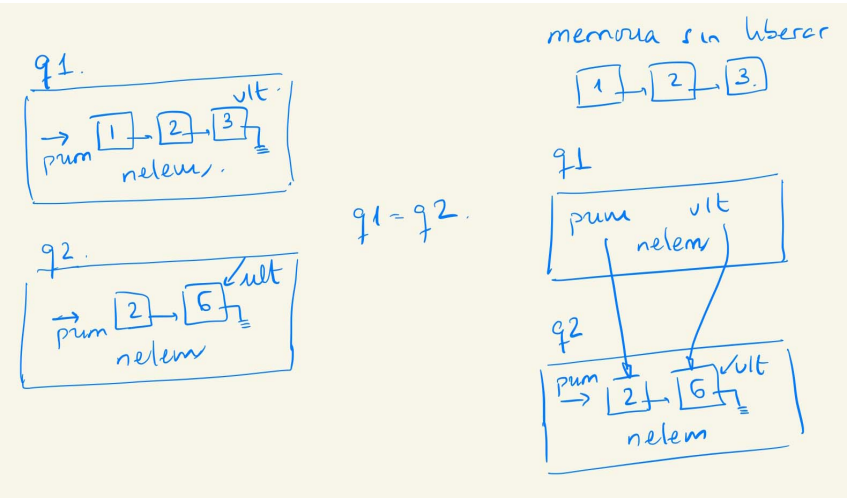
   Si, al ser la lista enlazada doble, se pueden añadir y eliminar elementos por el principio de la lista si se tiene un puntero al comienzo y se pueden añadir y eliminar elementos de la lista si se tiene un puntero al último nodo.

8. (1 punto) Sean `queue<int> q1, q2` dos colas implementadas con una lista enlazada simple y punteros al comienzo y final de la lista (implementación vista en clase). La cola `q1` tiene tres elementos 1,2,3 y la cola `q2` tiene dos elementos 2, 6. Indica cual es el resultado de la asignación q1 = q2 dibujando como quedan las listas enlazadas que representan las colas, despues de la asignación en cada uno de los siguientes casos:

   a) Utilizando el operador de asignación por defecto, es decir. en la clase `queue` no se implementa el operador de asignación.

   Respuesta:

   La asignación por defecto no hace copia de la memoria dinámica, solo copia los valores de los punteros. Tampoco libera la memoria de la cola a la que se le asigna el valor. Por lo tanto cuando se ejecuta la asignación ambas colas comparten los nodos en la memoria dinámica y los valores anteriores de la cola q1 quedan sin liberar.



b) Implementando un operador de asignación en la clase `queue` como el hecho en clase.

Respuesta:

La asignación realizada en la clase `queue_eda` primero libera la memoria que tenga la cola a la que se va a asignar el valor y a continuación crea nuevos nodos y copia los valores de la lista que se quiere asignar.