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, 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.top() << '\n';
p.push(q.top()); q.pop();
p.push(q.top()); q.pop();
std::cout << p.top() << '\n';
std::cout << q.top() << '\n';
```

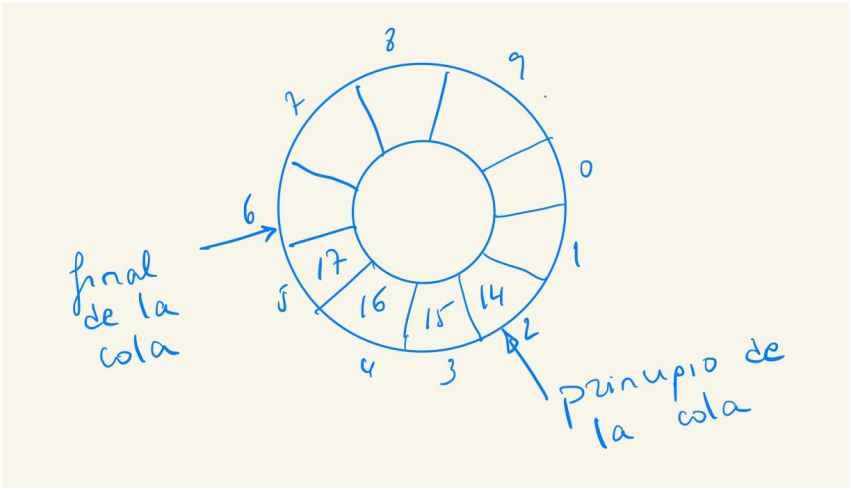
   Respuesta:

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

2. (2 puntos) Para obtener un coste constante en todas las operaciones de una cola (push, pop, front, empty) indica donde debe situarse el principio de la cola (parte por la que se eliminan los elementos) y el final (parte por la que se añaden los elementos) en cada una de estas implementaciones. Para ayudarte en la explicación dibuja la representación de una cola con 4 elementos en cada caso.

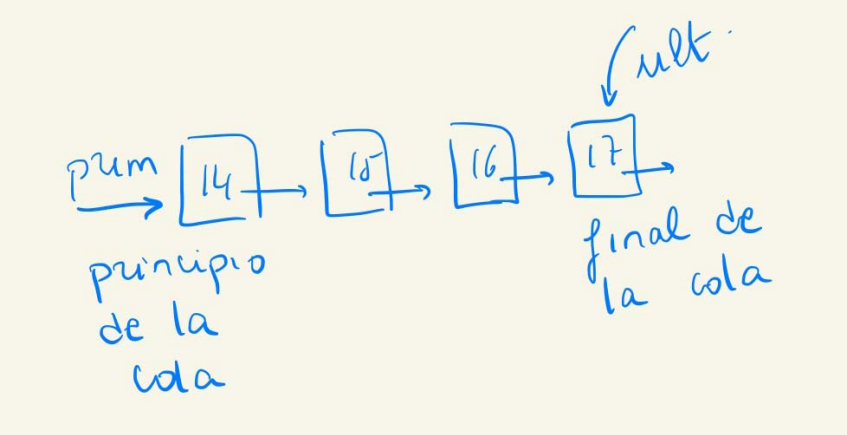
   Si la implementación se realiza con un array circular:

   Respuesta:



   Si la implementación se realiza con una lista enlazada simple:

   Respuesta:

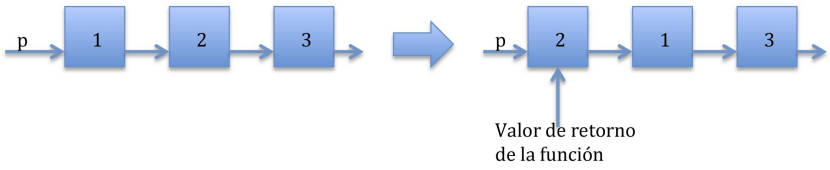


3. (1 punto) En la implementación de las colas por medio de un array se utiliza un *array circular*. Indica los motivos por los que se utiliza este tipo de array en lugar de un array plano.

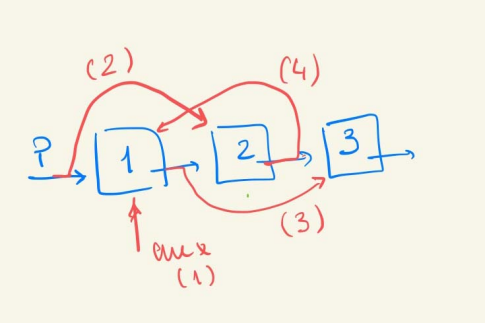
   Respuesta:

   Si se utiliza un array plano, para no tener que desplazar los elementos del vector en la operación `pop` o `push`, tendremos dos índices, uno en la parte izquierda del vector y otro en la parte derecha. El índice de la parte izquierda nos indica el elemento que se elimina de la cola (principio de la cola) y el índice de la parte derecha nos indica la posición por la que se añaden los elementos al vector. Con esta representación al ir añadiendo y eliminando elementos de la cola, esta se va desplazando en el vector. Cuando el índice de la derecha llegue al final del vector se producirá un error de cola llena, cuando puede haber muchos elementos libres en la parte izquierda del vector debido a los elementos que se han eliminado de la cola. Para evitar esto se utiliza el array circular, en el cual el siguiente al último elemento del vector es el primero. De esta forma solo se produce el error de cola llena cuando el vector está completo.

4. (1 punto) Completa el código de la siguiente función que intercambia dos nodos de una lista enlazada simple. El puntero que se recibe como parámetro apunta al primer nodo a intercambiar. Se garantiza que siempre hay al menos un nodo siguiente al apuntado por p. El valor de retorno es un puntero al nodo que queda en primera posición después de realizar el intercambio.



   Respuesta:




```
Nodo* intercambiar (Nodo* p){
        Nodo * aux = p;
         p = aux->sig;
         aux->sig = p->sig;
         p->sig = aux;
         return p;
}
```

5. (1 punto) ¿Qué hace la siguiente función cuando la llamamos con el primer nodo de una lista enlazada?

```
void fun(Nodo * prim) {
  if (prim != nullptr) {
    fun(prim->sig);
    cout << prim->elem << ' ';
  }
}
```

   a) Imprime todos los nodos de la lista

   b) Imprime todos los nodos de la lista en orden inverso

   c) Imprime nodos alternos de la lista

   d) Imprime nodos alternos de la lista en orden inverso

   Respuesta:

   b) Imprime todos los nodos de la lista en orden inverso

6. (1 punto) Dada una lista enlazada simple con un puntero al principio de la lista y otro puntero al final de la lista y cuyos valores están ordenados en orden creciente, indica si es posible aplicar el algoritmo de búsqueda binaria para buscar un elemento y justifica tu respuesta. Indica el coste que tendría el algoritmo.

   Respuesta:

   La búsqueda binaria no tiene sentido sobre una lista enlazada simple ya que el coste de buscar el elemento del medio de la lista es lineal en el número de elementos de la lista. Por lo tanto, solo la operación de buscar el elemento del medio tiene un coste del mismo orden de complejidad que una búsqueda secuencial sobre la lista.