ut3_pd1_respuestas

Ejercicio 1

Código:

```
Nuevo nodo otroNodo
otroNodo.siguiente ← nodo1
nodo2.siguiente ← nodo3
```

Situación inicial:

```
nodo1 → nodo2 → nodo3
```

Paso a paso:

1. `otroNodo.siguiente ← nodo1`
   Se crea un nuevo nodo que apunta al nodo1.
   Queda así:

   ```
   otroNodo → nodo1 → nodo2 → nodo3
   ```

2. `nodo2.siguiente ← nodo3`
   Esto no cambia nada porque nodo2 ya apuntaba a nodo3.

Respuesta correcta:
a) Inserta “otroNodo” en la lista, quedando como anterior a nodo1.

Ejercicio 2

Código:

```
Nuevo nodo otroNodo
otroNodo ← nodo1.siguiente
nodo1.siguiente ← nodo3
```

Situación inicial:

```
nodo1 → nodo2 → nodo3
```

Paso a paso:

1. `otroNodo ← nodo1.siguiente`
   otroNodo es una referencia a nodo2:

   ```
   otroNodo == nodo2
   ```

2. `nodo1.siguiente ← nodo3`
   nodo1 ahora apunta directamente a nodo3, eliminando nodo2 de la lista:

   ```
   nodo1 → nodo3
   ```

Respuesta correcta:
c) Elimina nodo2 de la lista.

Ejercicio 3

Código:

```
Nuevo nodo otroNodo
otroNodo.siguiente ← nodo1.siguiente
nodo1.siguiente ← otroNodo
```

Situación inicial:

```
nodo1 → nodo2 → nodo3
```

Paso a paso:

1. `otroNodo.siguiente ← nodo1.siguiente`
   nodo1.siguiente es nodo2, por lo tanto:

   ```
   otroNodo → nodo2
   ```

2. `nodo1.siguiente ← otroNodo`
   nodo1 ahora apunta a otroNodo:

```
nodo1 → otroNodo → nodo2 → nodo3
```

Respuesta correcta:
b) Inserta “otroNodo” en la lista, quedando entre nodo1 y nodo2.

Ejercicio 4

Código:

```
Nuevo nodo otroNodo
Nuevo nodo nodoActual
nodoActual ← primero
mientras nodoActual ≠ nulo hacer
  nodoActual ← nodoActual.siguiente
fin mientras
nodoActual.siguiente ← otroNodo
```

Problema:

Al finalizar el bucle, `nodoActual` es `nulo`. Luego se intenta hacer `nulo.siguiente ← otroNodo`, lo cual produce error.

Respuesta correcta:
d) El algoritmo está mal hecho, ya que dará siempre error en tiempo de ejecución.

Ejercicio 5

Código:

```
Nuevo nodo otroNodo
Nuevo nodo nodoActual
nodoActual ← primero
mientras nodoActual.siguiente ≠ nulo hacer
  nodoActual ← nodoActual.siguiente
fin mientras
nodoActual.siguiente ← otroNodo
```

Análisis:

- Recorre la lista hasta llegar al último nodo (cuyo siguiente es `nulo`).
- Le agrega `otroNodo` al final.

Pero si la lista está vacía (`primero` es `nulo`), la línea `nodoActual.siguiente` da error porque `nodoActual` es `nulo`.

Respuesta correcta:
c) El algoritmo está mal hecho, ya que dará error en tiempo de ejecución si la lista está vacía.

Ejercicio 6

a) Costo de memoria

Array:

- Reserva memoria fija al inicio.
- Si sobran espacios, se desperdicia memoria.
- Si falta espacio, hay que redimensionar (copia de datos).

Lista enlazada:

- Solo se asigna memoria cuando se agrega un nodo.
- Cada nodo requiere un puntero adicional, pero no hay desperdicio de memoria.

b) Consideraciones sobre cantidad de alumnos

Array:

- Requiere conocer un tamaño estimado.
- Es más eficiente en acceso aleatorio, pero inflexible ante cambios en la cantidad de alumnos.

Lista enlazada:

- Ideal para casos donde la cantidad de alumnos no está definida de antemano.
- Permite insertar y eliminar elementos dinámicamente sin redimensionar.

Conclusión:

Para un curso universitario con inscripciones cambiantes, como en el caso planteado, es más recomendable usar una lista enlazada por su flexibilidad y uso eficiente de la memoria.
