UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA

Guía de Ejercicios – Punteros

Algoritmos y **Estructura de datos**

Curso Ing. **Gustavo Schmidt**

Ejercicios Nivel Básico

1. **Intercambio de valores con aritmética (a = a + b, b = a - b, etc.)**
2. **Imprimir dirección de memoria de variables del tipo puntero y luego sus valores.**
3. **Ejecutar nuevamente el ejercicio anterior varias veces y comparar los valores.**
4. **Comparar valores de int y Integer y sus direcciones en memoria**
5. **Casteo entre int, float y double, mostrando los cambios de precisión**
6. **Uso de final en variables y qué pasa al intentar modificarla**
7. **Paso de variables primitivas a un método y ver si cambian dentro de él**
8. **Paso de un array a un método y ver si cambia su contenido**
9. **Crear un array y mostrar sus direcciones de cada elemento en memoria**
10. **Comparar == y .equals() en tipos primitivos y en String**

Ejercicios de Nivel Intermedio

11. **Intercambio de valores (swap) con variables locales**
    1. Implementa un método swap(int a, int b) y observa por qué no funciona en Java como en C++.
12. **Intercambio de valores usando un wrapper (clase contenedora)**
    1. Crea una clase NumberHolder { int value; } y usa un método swap(NumberHolder a, NumberHolder b) para ver cómo Java maneja referencias.
13. **Comparación de referencias en objetos vs valores primitivos**
    1. Declara dos objetos Integer con el mismo valor y verifica con == y .equals() si son iguales.
14. **Tamaño en memoria de tipos primitivos vs objetos**
    1. Usa Runtime.getRuntime().totalMemory() y freeMemory() para medir el impacto de crear variables primitivos vs objetos.
15. **Método recursivo simple (factorial) y su impacto en la memoria Stack**
    1. Implementa factorial(n) y observa cómo crece la pila con valores grandes.
16. **Vector de enteros y su ubicación en Heap**
    1. Declara un int[] y observa que aunque se declare en un método, sigue existiendo en Heap.
17. **Asignación de referencias de objetos y su efecto en la memoria**
    1. Declara dos objetos de una misma clase y asigna uno al otro.

¿Qué ocurre con la referencia anterior?

18. **Paso de un array a un método y su modificación dentro del método**
    1. Muestra cómo los cambios en un array dentro de un método afectan al original.
19. **Ciclo de vida de un objeto y el recolector de basura**
    1. Crea y pierde referencias a objetos para que el Garbage Collector los elimine.
20. **Copiar un array de forma profunda vs superficial**

- Usa System.arraycopy() y luego implementa una copia manual profunda.

Ejercicios de Nivel Avanzado

21. **Uso de WeakReference y SoftReference para administrar memoria**
    1. Crea referencias débiles y suaves y observa cómo el recolector de basura las maneja.
22. **Crear una cache simple**
    1. Implementa una caché donde un vector levante los datos de un archivos y compara los tiempos en la lectura del archivo y la del vector.
23. **Clona profunda y superficial de objetos**
    1. Implementa un vector y clona el mismo, en una oportunidad copiando el puntero y en otra copiando el 100% de los datos y compara los tiempos.
24. **Manejo de grandes volúmenes de datos en arrays**
    1. Carga datos en memoria y compara velocidad en tamaños chicos de vectores y en tramaños muy grandes.

25. **Simulación de OutOfMemoryError controlada**
    1. Fuerza un OutOfMemoryError creando muchos objetos y analiza el log de la JVM.
26. **Análisis de memoria en estructuras de datos personalizadas**
    1. Implementa una estructura similar a un vector y mide su consumo de memoria.
27. **Demostrar la diferencia entre `++i` y `i++` en evaluaciones**

    Usa `int x = 5; int y = x++ + ++x;` e imprime valores paso a paso.

28. **Medir el impacto en memoria de un array grande**

    Crea un `int[1_000_000]`, mide la memoria antes y después con `Runtime.getRuntime()`.