UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ciencias


# Introducción a las Ciencias de la Computación

*Práctica 8: Instrucciones de control II*

Profesora:
Amparo López Gaona
Ayudante:
Víctor Emiliano Cruz Hernández
Ayudante de Laboratorio:
Kevin Jair Torres Valencia

# Objetivos

El objetivo de esta práctica es que el alumno aprenda el uso de ciclos como instrucciones de iteración en un programa y refuerce sus conocimientos acerca de arreglos de datos de tipo primitivo. Ejercitando la inicialización, llenado y consulta de arreglos, así como el trabajo con instrucciones de iteración.

# Introducción

La recursión es un concepto en programación que implica que una función se llame a sí misma directa o indirectamente. En esencia, es una técnica que divide un problema en casos más simples y similares, resolviendo cada caso base directamente y utilizando el mismo proceso para resolver instancias más complejas del problema. La recursión suele utilizar una estructura de control basada en la repetición del mismo bloque de código con datos modificados en cada llamada, hasta alcanzar una condición base que termina la recursión. Este enfoque es útil para resolver problemas que se pueden descomponer en subproblemas más pequeños y similares.

Una instrucción de iteración permite la ejecución repetida de instrucciones contenidas en un bloque. Java proporciona tres tipos de instrucciones iterativas.

- La instrucción `do-while` tiene la siguiente sintaxis:

  ```
  do {
      // Cuerpo de la instruccion.
  } while (condicion);
  ```

  Con la instrucción `do` se ejecuta el bloque de instrucciones que contiene entre llaves mientras la condición evaluada al final del mismo sea verdadera. En cuanto el resultado de la evaluación de la condición sea falsa se termina la instrucción do. Es decir, de antemano no se puede conocer el número de iteraciones que se realizarán y las instrucciones del bloque se realizan al menos una vez.

- La sintaxis de la instrucción `while` es la siguiente:

  ```
  while (condicion) {
      // Cuerpo de la instruccion.
  }
  ```

  y su semántica es realizar el bloque de instrucciones mientras la evaluación de la condición sea verdadera. A diferencia de la instrucción `do`, primero se evalúa la condición y luego se realiza el bloque de instrucciones, con lo cual cabe la posibilidad de que la primera evaluación de la condición dé el valor `false` y por lo tanto las instrucciones del cuerpo no se ejecuten ni siquiera una vez.

- La sintaxis de la instrucción `for` es la siguiente:

```
for (inicializacion; condicion; actualizacion) {
    // Cuerpo de las instruccion.
}
```

La forma de trabajar de esta instrucción es ejecutar la expresión de inicialización una vez, luego evaluar la condición y si devuelve true se realiza el cuerpo, seguido de la actualización. Se repiten estos pasos desde la evaluación de la condición hasta que la condición tenga valor `false`.

- La sintaxis de la instrucción `for-each` es la siguiente:

```
for (tipo elemento : arreglo/cadena) {
    // Cuerpo de la instruccion.
}
```

El bucle `for-each` en Java es una estructura simplificada (envoltura de la estructura `for`) que se utiliza para recorrer elementos de una colección o arreglo. En lugar de utilizar un contador para iterar sobre los elementos y acceder a ellos por índice, el bucle for-each permite recorrer los elementos de manera más directa y legible. Utiliza una sintaxis que especifica el tipo de elementos y una variable que representa cada elemento en la colección. Esto hace que el código sea más claro y conciso al eliminar la necesidad de manejar índices manualmente.

# Desarrollo

**1.** Dado un texto compuesto solo por letras y espacios, escribe un programa que cuente cuántas palabras son palíndromos (se leen igual de izquierda a derecha que de derecha a izquierda).
**Restricciones:**

- No se pueden usar estructuras de datos como `arreglos, ArrayList, StringBuilder, etc..`
- Solo se pueden utilizar estructuras de control (`for, while, do-while`)
- Se deben ignorar los espacios entre palabras y considerar solo las letras.
- No se permite el uso de métodos de `reverse(), split(), chartAt()` ni `substring()`

Ejemplo de ejecución:

- Entrada: Texto: `"oso ana sol radar luz"`
- Salida Numero de palabras palindromas: `3`

**2.** El siguiente código encripta claves:

```
public class cifrado01 {
    private long cifrado;
    private final int semilla = 79;
    private final String diccionario = "abefimnoprsuv";

    /**
     * @param clave Clave a cifrar.
     */
    public cifrado01(String clave){
        cifrado = recursion(semilla*13 + diccionario.indexOf(clave.charAt(0))
            ↪ , clave.substring(1));
    }

    /* Recursion sobre "clave" y encripta mediante potencias de 13. */
    private long recursion(long acc, String resto) {
        if(resto.length() == 1) return acc*13 + diccionario.indexOf(resto.
            ↪ charAt(0));
        return recursion(13*acc + diccionario.indexOf(resto.charAt(0)), resto
            ↪ .substring(1));
    }

    /**
     * @return long - cifrado. Devuelve la palabra "clave" cifrada.
     */
    public long obtenerCifrado() { return cifrado; }
}
```

Tú misión en este problema es reescribir el código de manera iterativa. Es decir, el código que escribas debe mantener el funcionamiento pero debe ser escrito con instrucciones iterativas y ser documentado correctamente.

Por último, escribe de manera detallada como funciona tu código y porque se te ocurrió escribirlo de esta manera.

**Recomendación:** Realicen una ejecución de la función que encripta una palabra con ”perseverar”

**4.** Escribe un programa que realice un ordenamiento lexicográfico mediante el algoritmo `InsertionSort`. Tú programa debe recibir una cadena bajo cualquier orden y debe devolver la cadena ordenada. Considera solo 26 letras del abecedario y 10 dígitos. Puedes dar por hecho que la cadena de entrada no contiene carácteres especiales.

Consideraciones:

1. Escribe una versión recursiva.
2. Escribe una versión iterativa.

# Formato de Entrega

1. Las prácticas serán entregadas en parejas y solo un integrante debe entregarla.
2. Cada práctica (sus archivos y directorios) deberá estar contenida en un directorio llamado `EquipoX_pY`, donde:
   (a) X es el número de equipo correspondiente.
   (b) Y es el número de la práctica.

   Por ejemplo: `Equipo01_p01`
3. NO incluir los archivos .class dentro del directorio a entregar.
4. Los archivos de código fuente deben estar documentados.
5. Se pueden discutir y resolver dudas entre los integrantes del grupo. Pero cualquier práctica plagiada total o parcialmente será penalizada con cero para los involucrados.
6. La práctica se debe entregar en el apartado de Github Classroom correspondiente.
7. El horario y día de entrega se acordará en la clase de laboratorio y no deberá sobrepasar 2 clases de laboratorio.