

UNIVERSIDAD NACIONAL AUTÓNOMA DE
MÉXICO
Facultad de Ciencias



Introducción a las Ciencias de la Computación

Práctica 07: Instrucciones secuenciales II.

Profesora: Amparo López Gaona
Ayudante: Ramsés Antonio López Soto
Ayudante: Adrián Aguilera Moreno

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. 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 caracteres especiales.

Consideraciones:

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

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 decifrar la clave de 10 caracteres apartir del valor numérico que devuelve el método constructor anterior.

Obs. En este problema eres libre de usar recursión o iteración.

3. Un palindromo es una cadena que se puede leer de izquierda a derecha y de derecha a izquierda de la misma manera, e.g. 10001, ana, MO1M1OM.

Deseamos que a partir de una cadena palindroma construyamos una nueva cadena que cumpla con los siguientes puntos:

1. La nueva cadena es palíndroma.
2. La nueva cadena debe ser la menor posible respecto a un orden lexicográfico.
E.g. `naan` \mapsto `anna`.

Obs. Considera que, si no existe una cadena palíndroma menor a la cadena original, entonces tú programa debe devolver la cadena original.

Crea un programa que resuelva este problema usando:

1. Recursión.
2. `do-while`.
3. `while`.

Hint. Usa el programa que generaste a partir del problema 1. Este no necesita estar implementado con alguna de las instrucciones mencionadas.

4. Escribe un programa que reciba un arreglo y devuelva el arreglo invertido. Para mejorar la visualización, debes implementar un método que imprima cualquier arreglo en su formato usual (formato de idea).

Escribe las siguientes versiones de tú programa:

1. Recursiva.
2. Utilizando la instrucción `for`.
3. Usando la envoltura `for-each`.
4. Escribe una versión basada en la instrucción `while`.
5. Utiliza la instrucción `do-while`.

Formato de Entrega

1. Las prácticas serán entregadas de forma individual.
2. Cada práctica (sus archivos y directorios) deberá estar contenida en un directorio llamado `apellidoPaterno_nombre_pX`, donde X es el número de la práctica.
Por ejemplo: `aguilera_adrian_p7`
3. NO incluir los archivos `.class` dentro de la carpeta.
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 subir al Github Classroom correspondiente.
7. La entrega en classroom debe contener el link HTTPS y SSH de su repositorio y es lo único que se debe entregar.
8. El horario y día de entrega se acordará en la clase de laboratorio y no deberá sobrepasar 2 clases de laboratorio.