Programación I
Ejercicios de subprogramas

Pablo Garaizar Sagarminaga
Borja Sanz Urquijo

Facultad de Ingeniería

Estrellas

- Programa la función imprimeEstrellas(n):

  imprimeEstrellas(5);

  *****

Estrellas

- Programa la función imprimeEstrellas(desde, hasta):

```
imprimeEstrellas(2, 5);
**
***
****
*****
```

Triángulo

- Programa la función imprimeTriangulo(altura) que usa imprimeEstrellas(n):

```
imprimeTriangulo(5);
*
**
***
****
*****
```

Pirámide

- Programa la función imprimePiramide(altura) que usa imprimeEspacios(n) y imprimeEstrellas(n):

```
imprimePiramide(5);
        *
       ***
      *****
     *******
    *********
```

Primos

- Programa la función esPrimo(n) que devuelve True si n es primo y False si no lo es.

Factorial

- Programa la función factorial(n) que devuelve el factorial de n.

Fibonacci

- Programa la función que calcule el valor enésimo de la serie de Fibonacci:

  fb(0) = 0

  fb(1) = 1

  fb(n) = fb(n-1)+fb(n-2).

  n = fibonacci(3)

  // n = fb(2)+fb(1) → n = fb(1) + fb(0) + fb(1)

  // n = 1 + 0 + 1 → n = 2

Calculadora

- Programa una calculadora que use las funciones sumar(op1, op2), double restar(op1, op2), double multiplicar(op1, op2), double dividir(op1, op2) y menu():

```
1. Sumar
2. Restar
3. Multiplicar
4. Dividir
Opcion? 1
Operando 1: 10
Operando 2: 21
Resultado: 31
```

Binario a decimal

- Programa una función que reciba un string con un número binario y devuelva el número entero correspondiente:

  n = binarioADecimal(“1011”) # n vale 11

Decimal a binario

- Programa una función que reciba un entero y devuelva el número binario correspondiente en un string:

```
binario = decimalABinario(6)
# binario vale “110”
```

Dos y cuatro

- Programa una función que reciba una lista de enteros y devuelva True si contiene un 2 o un 4, pero no ambos:
  - dosYCuatro([1,2,3]) # True
  - dosYCuatro([3,4,5]) # True
  - dosYCuatro([1,3,5]) # False
  - dosYCuatro([1,2,3,4]) # False

Mayor array

- Programa una función que reciba dos listas de enteros del mismo tamaño y devuelva True si cada elemento de la 1a lista es mayor que el correspondiente elemento en la 2a lista:
  - mayor([1,2,3], [0,1,2]) # True
  - mayor([7,5,3], [6,3,0]) # True
  - mayor([1,0,3], [0,1,2]) # False

Terminar igual

- Programa una función que reciba dos listas de enteros y un entero n. Devolverá True si la 1a lista termina igual que la 2a lista en los n últimos elementos:
  - terminarIgual([1,2,3,4], [0,1,3,4], 2) # True
  - terminarIgual([1,2,3,4], [0,2,3,4], 3) # True
  - terminarIgual([1,2,2,3,4], [0,1,2,3,4], 4) # False

Dos junto a dos

- Programa una función que reciba una lista de enteros y devuelva True si hay un 2 junto a otro 2, o false en caso contrario:
  - dosJuntoADos([1,2,2,4]) # True
  - dosJuntoADos([1,2,3,4]) # False
  - dosJuntoADos([1,2,3,2]) # False

Mayor diferencia

- Programa una función que reciba una lista de enteros y devuelva la diferencia entre el número más alto y el número más bajo:
  - mayorDiferencia([5,2,3,2,6,3,4]) # 6-2 = 4
  - mayorDiferencia([1,1,1,4]) # 4-1 = 3

Rima

- Programa una función que reciba dos string y un entero n y diga si riman, es decir, si son iguales los n últimos caracteres:
  - rima(“carcasa”, “pasa”, 2) # True
  - rima(“carcasa”, “pisa”, 2) # True
  - rima(“carcasa”, “pisa”, 3) # False

Suma dígitos

- Programa una función que reciba un string y devuelva la suma de los dígitos que contiene (cero si no contiene ninguno):
  - sumaDigitos(“c1a2sa”) # 3
  - sumaDigitos(“c1a2r4c5a6”) # 18
  - sumaDigitos(“casa”) // 0

Parsea entero

- Programa una función que reciba un string y devuelva el entero correspondiente de juntar todos los dígitos que contiene (cero si no contiene ninguno):
  - parseaEntero(“c1a2sa”) # 12
  - parseaEntero(“c1a2r4c5a6”) # 12456
  - parseaEntero(“casa”) # 0

Borra letras

- Programa una función que reciba dos string y devuelva el primero de ellos después de haberle quitado todos los caracteres del segundo:
  - borraLetras(“murcielago”, “aeiou”) # “mrclg”
  - borraLetras(“cama”, “coche”) # “ama”
  - borraLetras(“carcasa”, “ca”) # “rs”

Rachas

- Programa una función que reciba una lista de enteros y devuelva el número de rachas que contiene (dos o más veces el mismo número seguido):
  - rachas([1,2,3,4,5,6]) # 0
  - rachas([1,1,3,4,5,6]) # 1
  - rachas([1,1,3,4,4,6]) # 2
  - rachas([1,1,1,1,5,6]) # 1
  - rachas([1,1,3,3,5,6,6]) # 3

Partible

- Programa una función que reciba una lista de enteros y devuelva True si existe una manera de dividir en dos partes la lista tal que los números de una parte sumen lo mismo que los de la otra:
  - partible([1,1]) # parto por la mitad 1 = 1 → True
  - partible({1,1,2]) # parto por el 2º, 1+1 = 2 → True
  - partible([1,2,1,4,1]) # parto por el 3º, 1+2+1 = 4 +1 → True
  - partible([7,5,1,4,1]) # False

Ahorcado

- Juego del ahorcado, con las siguientes funciones:
  - elegirPalabra() → elegir aleatoriamente una palabra de una lista de palabras
  - pedirLetra() → pedir letra al jugador
  - comprobarLetra(palabra, letra) → comprobar si la letra pertenece a la palabra (True) o no (False)
  - pintarAhorcado() → dibujar el estado actual

Ahorcado

- java Ahorcado

```
Palabra: _ _ _ _ _
Letra? m
¡La letra m sí pertenece a la palabra!
Palabra: M _ _ _ _
Letras correctas: M
Letra? x
¡La letra x NO pertenece a la palabra!
Palabra: M _ _ _ _
Letras correctas: M
Letras incorrectas: X



___
Letra? V
¡La letra x NO pertenece a la palabra!
Palabra: M _ _ _ _
Letras correctas: M
Letras incorrectas: X V
 |
 |
_|_
```

Tres en raya

- Juego en el que habrá una variable booleana para decidir a quién le toca jugar (CPU o ser humano) y los siguientes métodos:
  - printBoard
  - selectCellCPU → ¿cómo hacemos la IA?
  - selectCellHuman
  - checkWinner

Tres en raya

- java TicTacToe

```
Mi turno:
[ ][ ][ ]
[ ][X][ ]
[ ][ ][ ]
Tu turno. Introduce fila y columna: 1 1
[O][ ][ ]
[ ][X][ ]
[ ][ ][ ]
Mi turno:
[O][ ][ ]
[ ][X][ ]
[ ][X][ ]
Tu turno. Introduce fila y columna: 1 2
[O][O][ ]
[ ][X][ ]
[ ][X][ ]
```

Hundir la flota

- Juego en el que habrá 1 tablero de 10x10 para cada jugador donde se colocarán de forma vertical u horizontal los siguientes barcos:
  - 1 portaaviones (5 casillas).
  - 2 destructores (4 casillas).
  - 3 buques (3 casillas).
  - 5 lanchas (1 casilla).
  - Los barcos propios tendrán que estar al menos a una casilla de distancia entre sí.

Hundir la flota

- Implementación:
  - 4 listas de juego:
    - boardShipsCPU, boardShotsCPU.
    - boardShipsHuman, boardShotsHuman.
  - Funciones:
    - placeShipsCPU
    - placeShipsHuman
    - shootCPU
    - shootHuman
    - printBoardsHuman
    - checkWinner

Switch off!

- Juego en el que se dispone una matriz de 8x8 con luces encendidas y apagadas. Cada vez que se elige una celda, se cambia de estado las luces de arriba, abajo, izquierda y derecha (si las hubiera). El objetivo es apagarlas todas.
  - Sería bueno tener una lista de niveles, empezando con el más fácil y aumentando la complejidad.

Switch off!

- java Switch

```
   1  2  3  4  5  6  7  8
1 [ ][ ][ ][ ][ ][ ][ ][ ]
2 [ ][ ][ ][ ][ ][ ][ ][ ]
3 [ ][ ][ ][*][ ][ ][ ][ ]
4 [ ][ ][*][ ][*][ ][ ][ ]
5 [ ][ ][ ][*][ ][ ][ ][ ]
6 [ ][ ][ ][ ][ ][ ][ ][ ]
7 [ ][ ][ ][ ][ ][ ][ ][ ]
8 [ ][ ][ ][ ][ ][ ][ ][ ]
Fila y columna? 4 4
   1  2  3  4  5  6  7  8
1 [ ][ ][ ][ ][ ][ ][ ][ ]
2 [ ][ ][ ][ ][ ][ ][ ][ ]
3 [ ][ ][ ][ ][ ][ ][ ][ ]
4 [ ][ ][ ][ ][ ][ ][ ][ ]
5 [ ][ ][ ][ ][ ][ ][ ][ ]
6 [ ][ ][ ][ ][ ][ ][ ][ ]
7 [ ][ ][ ][ ][ ][ ][ ][ ]
8 [ ][ ][ ][ ][ ][ ][ ][ ]
¡Lograste apagar todas las luces!
```

El juego de la vida de Conway

- Definir un patrón inicial de celdas vivas y muertas en una lista de 20x20. En un bucle infinito, calcular las siguientes generaciones:
  - Si una celda está viva y tiene menos de 2 vecinas vivas, muere.
  - Si una celda está viva y tiene más de 3 vecinas, muere de superpoblación.
  - Si una celda está muerta y tiene exactamente 3 vecinas vivas, revive.

El juego de la vida de Conway