



## Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

# Laboratorios de computación salas A y B

Edgar Tista Garcia

*Profesor:*

Estructuras de Datos y Algoritmos I

*Asignatura:*

2

*Grupo:*

11

*No. de práctica(s):*

Álvarez López Carlos Manuel

*Integrante(s):*

2

*No. de lista o brigada:*

2023-2

*Semestre:*

Martes 6 del 2023

*Fecha de entrega:*

*Observaciones:*

CALIFICACIÓN: \_\_\_\_\_

**OBJETIVO:** Aplicar el concepto de recursividad para la solución de problemas

**OBJETIVO DE CLASE:** Conocer algunas implementaciones de funciones recursivas para familiarizarse con su ejecución

## **DESARROLLO DE LA PRÁCTICA:**

1.- ¿Qué es la recursividad?, ¿cómo se define una función recursiva?

La recursividad consiste en la capacidad de determinar un elemento o resolver un problema dividiendo el problema en subproblemas más simples.

Una función recursiva es aquella en la que la función se ejecuta a sí misma. Estas suelen constar de un caso base y un caso recursivo que es el encargado de separar el problema más complejo en problemas más simples. El caso base funciona como una condición para detener la recursividad y detener la función.

```
funcionRecursiva(n){  
    //código, caso base  
    return funcionRecursiva(n);  
}
```

2.- Codifica el programa e indica qué hace

```
def mystery(a, b):  
    if b == 0:  
        return a  
    elif b > 0:  
        return mystery(a + 1, b - 1)  
    else:  
        return mystery(a - 1, b + 1)
```

La función `mystery` recibe 2 parámetros, siendo “a” y “b”. La función se compone de una estructura `if-elif-else`, en la cual la condición del `if` es si “b” es igual a 0, en cuyo caso retornara “a”, siendo este el caso base. Si la condición del `if` no se cumple, se pasa a `elif`, que evalúa si “b” es mayor que 0, a lo cual se retornará lo que retorne la función `mystery(a+1,b-1)`, siendo este uno de los casos recursivos, ya que ejecuta la misma función pero con un incremento y decremento a los parámetros para acercarnos al caso base. En caso de que ninguna de las condiciones anteriores se cumpla, nos indica que b es menor a 0, por lo que la función retorna lo que retorne la función `mystery(a-1,b+1)`, acercándonos así mismo al caso base, siendo que “b” sea igual a 0. Esta función también se puede interpretar como que se le suma “a” a “b”.

```
Guadalupe49:P11 edaI02a
a = 2, b = 2
4
Guadalupe49:P11 edaI02a
a = 2, b = -2
0
Guadalupe49:P11 edaI02a
a = 2, b = 4
6
Guadalupe49:P11 edaI02a
a = 2, b = -4
-2
Guadalupe49:P11 edaI02a
a = -3, b = -4
-7
```

3.- Codifica, compila y ejecuta el siguiente programa e indica qué hace

```
def funcion3(L,n)
    if n == 1:
        return L[0]
    else:
        return L[0] + funcion3(L[1:],n-1)
lista1=[1,2,3,4,5,6,7,8]
lista2=funcion3(lista1,n)
print(lista2)
```

En esta función se reciben 2 parámetros, L, siendo una lista, y n, que se trata como la posición de lista. Esta función retorna la suma de los n primeros elementos de la lista L. Para esto se tiene el caso base  $n==1$ , siendo si solo se va a tomar el primer elemento de la lista, en caso de que n sea diferente a 1, la función retorna la suma del primer elemento de la lista recibida más lo que retorne la `funcion3(L[1:],n-1)`, siendo esto la misma función pero eliminando el primer elemento de la lista recibida (que ya se está sumando) y disminuyendo n, para que cuando se llame a la función vuelva a tomar el primer elemento

```
n = 1
1
Guadalup
n = 2
3
Guadalup
n = 3
6
Guadalup
n = 4
10
Guadalup
n = 5
15
```

4.- Codifica y compila el siguiente programa, explica cómo se resuelve el problema de las Torres de Hanoi con este programa

```
def move(n, x, y, z):
    if n == 1:
        print ('move', x, 'to', y)
    else:
        move(n-1, x, z, y);
        print ('move', x, 'to', y)
        move(n-1, z, y, x);
move(10, "A", "B", "C")
```

En este programa se declara la función `move(n,x,y,z)` que imprime las instrucciones para resolver el problema de las torres de Hanoi con `n` discos en la primera torre. Los parámetros se podrían interpretar como a “x” como la torre de origen, “y” como la torre destino y “z” como la torre auxiliar para mover los discos.

Esta función tiene como caso base a `n==1`, indicando que solo hay un disco en la torre origen por lo que solo se tendría que pasar a la torre destino. En caso de que sea distinto a 1, se llama a la función `move()` reduciendo en 1 “n” e invirtiendo las posiciones de la torre destino y la torre auxiliar, este proceso se repite hasta que se llega al caso base, pero sin imprimir ninguna instrucción, solo habiendo cambio en la

disposición de las torres si la  $n$  inicial es par, moviendo el primer disco a la torre destino, a diferencia de que si fuera impar, que movería el disco a la torre auxiliar. Seguido de esto se mueve el siguiente disco a la torre que no se ocupó con el disco anterior; seguido de esto se tiene la segunda llamada recursiva de move, en la cual se realiza un decremento de  $n$  y la torre auxiliar y la torre origen invierten posiciones, esto se repite hasta que solo quede un disco en la torre de origen, y se pase directamente ese disco a la torre destino, mientras que todos los otros discos están en la torre auxiliar.

5.- ¿Cuáles consideras las principales ventajas y desventajas de la recursividad (2 y 2)

#### Ventajas

- Las funciones recursivas al tener un caso base, tienen una sintaxis fácil de analizar.
- Permite reusar el código, en caso donde se divida un problema en subproblemas, se puede reutilizar la misma función.

#### Desventajas

- Ocupa más memoria que una estructura iterativa.
- Su tiempo de ejecución aumenta demasiado en relación a la entrada.

6.- ¿Qué es el retroceso recursivo? Explica si se aplica siempre que se tiene una función recursiva o no y por qué

Se trata de realizar las operaciones siguiendo la forma bottom-up, la cual se toman en cuenta los procesos desde el último módulo introducido hasta el primero; en otras palabras las llamadas recursivas se van resolviendo desde la última que se hizo, hasta la primera.

Normalmente debido a la naturaleza recursiva y de división en subproblemas que suelen tener los problemas en los cuales se utiliza la recursividad para la solución, suele ser más sencillo y eficaz la aplicación del retroceso recursivo. Aun así, según la naturaleza del problema, puede que el enfoque bottom-up no sea el ideal, debido al surgimiento de cálculos redundantes o complejidad innecesaria en el código.

### **Actividad 2.- Realiza los siguientes ejercicios relacionados con programas que contienen funciones recursivas**

#### **2.1 Elabora un programa en lenguaje c para evaluar y probar la siguiente función**

```

cat(int num) {
    if(num%2==0)
        num--;
    if (num > 0){
        cat(num-2);
        printf("%d \n",num);
    }
}

```

La función tiene un condicional que hace que si la entrada es par, a esta se le realiza un decremento, volviéndola impar; seguido de esto, si la entrada es mayor a 0, se vuelve a llamar a la función cat() pero dando como parámetro la entrada de la llamada actual menos dos, para finalmente imprimir la entrada.

Lo que resulta es que en pantalla se imprime una sucesión de números impares desde 1 hasta el menor número impar más próximo a la entrada.

```

1
3
5
7

```

### **Invierte el orden de las instrucciones del segundo if, explica el porqué de los resultados obtenidos**

La razón por la que se invierte la sucesión de números impares es por que antes de que se realice la llamada recursiva, se imprime la entrada de la función, y una vez se hace la llamada con la entrada decrementada, se vuelve a realizar lo mismo, imprimir la entrada y luego realizar la llamada; a diferencia de el caso anterior cuando primero se realizaba la llamada ocasionando que la entrada se redujera hasta ser menor que 0 y que comenzará el retroceso.

```

7
5
3
1

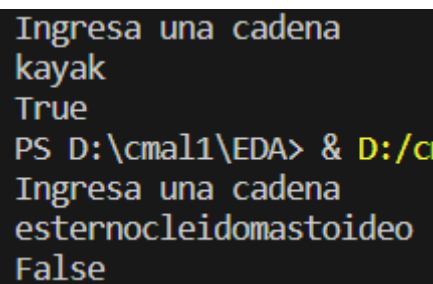
```

## 2.2 Indica qué hace el siguiente programa (Python)

```
def mystery(S):
    N = S.split()
    N = ''.join(N)
    if len(N) == 1 or len(N) == 0:
        return True
    else:
        if N[0] == N[-1] and mystery(N[1:-1]):
            return True
        else:
            return False
test = input("Ingresa una cadena \n")
print(mystery(test))
```

Esta función recursiva recibe una cadena como parámetro y devuelve True si es un palíndromo y False en caso contrario. Primero la función iguala la variable N a partir de la cadena recibida, crea una lista con las palabras de la cadena con el método split(), para después con el método join() juntarlas en una cadena sin los espacios de la cadena original. Seguido de esto, la función comprueba si la longitud de la cadena es de 1 o 0, dado que estas cadenas automáticamente serán palíndromos, y se devolverá True.

Si la longitud de la cadena es mayor a 1, si el primer y el último carácter de la cadena son iguales y la llamada de la función mystery() usando como parámetro la misma cadena pero sin tomar el primer ni el último carácter, es verdadero entonces la función retorna True, y para el caso contrario regresa False. La segunda parte de la expresión booleana donde se llama recursivamente a la función, es la encargada de ir verificando cada par de caracteres de la cadena hasta que se llegue al caso base y después ir resolviendo las llamadas recursivas que se han hecho, siendo que si en algún momento algún carácter no coincidió, entonces el resultado de la llamada principal será False.



```
Ingresa una cadena
kayak
True
PS D:\cma11\EDA> & D:/c
Ingresa una cadena
esternocleidomastoideo
False
```

### 3.- En clase vimos una función recursiva para sumar dos números, ahora escribe una función recursiva similar para multiplicar a y b

La función multiplicar recibe 2 parámetros que serán números enteros. Como caso base se tiene si b igual a 0, en cuyo caso se retornará 0. Si b es mayor que 0, se retornará la entrada a sumado a lo que retorne la función multiplicar(a,b-1) siendo que la función volverá a caer en el mismo caso hasta llegar al caso base, sumando de esta manera la variable "a" el mismo número de "b". Para cuando la variable b sea negativa, entonces se retorna el negativo de lo que retorne la función multiplicar(a,-b) lo cual nos llevará al mismo resultado que si "b" fuera positivo pero al multiplicarlo por un menos, nos dará el resultado deseado.

Para esta función se vio a la multiplicación como la suma de "a" más "a", "b" veces. Siendo que para el caso base, si un número es 0, entonces directamente se devolverá 0, y de manera similar que la función de suma que hicimos en clase, esta función va sumando cada llamada, siendo que el número de llamadas hechas depende del valor de b.

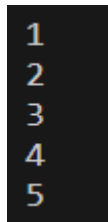
```
PS D:\cma11\EDA> 5
2
Resultado = 10
PS D:\cma11\EDA> -7
8
Resultado = -56
PS D:\cma11\EDA> -4
-3
Resultado = 12
PS D:\cma11\EDA> 0
7
Resultado = 0
```



#### **4.- Elabora una función recursiva para imprimir el contenido de una lista ligada simple (puedes probarla con las bibliotecas de lista ligada elaborada en la práctica 7)**

La función `imprimirLista()` recibe el nodo de una lista, a partir del cual se quiere empezar a imprimir. Esta función imprimirá los valores de los nodos de la lista hasta llegar al último. Primero imprime el valor del nodo recibido y mediante un `if` si la propiedad `next` del nodo es distinta de `NULL`, entonces se llama a la función `imprimirLista()` usando como parámetro el nodo siguiente del nodo de la llamada actual. De este modo la función va imprimiendo y llamándose a sí misma hasta llegar al final de la lista, siendo que se encuentra con que el siguiente nodo es `NULL`.

Para la comprobación de su funcionamiento se utilizaron las bibliotecas de la práctica 7, y una lista cuyos nodos tienen los valores de 1,2,3,4,5 respectivamente.



```
1
2
3
4
5
```

#### **Conclusión**

Ya habiendo completado los ejercicios de la práctica en su totalidad, se puede afirmar que los objetivos de la clase fueron cumplidos ya que durante los ejercicios se observaron algunas implementaciones de recursividad a diversas situaciones, además de que se implementaron para crear soluciones a algunos problemas. Tanto los ejercicios de análisis así como los de crear nuestras propias funciones fueron importantes para comprender las implicaciones de la recursividad en su implementación.

La recursividad es una herramienta que resulta bastante útil según la situación, dándonos una eficaz manera de solucionar problemas del tipo divide y vencerás, o problemas que cuentan naturalmente con propiedades recursivas. Si son correctamente implementadas nos pueden dar un código fácil de leer y eficaz al solucionar el problema, aunque si el problema no es manejado apropiadamente o no es tan compatible con la recursividad, podemos terminar con un código complejo e ineficaz en términos de memoria y tiempo.

## References

Definicion ABC. (n.d.). *Definición de Recursividad » Concepto en DefinicionABC*.

Definición ABC. Retrieved May 16, 2023, from

<https://www.definicionabc.com/comunicacion/recursividad.php>

Educative. (n.d.). *What is recursion?* Educative.io. Retrieved May 16, 2023, from

<https://www.educative.io/answers/what-is-recursion>