	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: TISTA GARCÍA EDGAR

Asignatura: ESTRUCTURAS DE DATOS Y ALGORITMOS I

Grupo: 1

No de Práctica(s): 12

Integrante(s): GÓMEZ LUNA ALEJANDRO

No. de Equipo de cómputo empleado: 42

No. de Lista o Brigada: 14

Semestre: 2019-2

Fecha de entrega: 15/Mayo/2019

Observaciones:

CALIFICACIÓN: _____

- Objetivo de la práctica
  APLICAR EL CONCEPTO DE RECURSIVIDAD PARA LA SOLUCIÓN DE PROBLEMAS

- Desarrollo
  Actividad 1: En la guía se encuentra la definición de recursividad, así como las reglas que esta debe de cumplir al momento de ser empleada para que sea llevada a cabo correctamente.
  Una vez planteado esto, se proceden con algunos ejemplos que nos permiten visualizar de mejor manera cómo y cuándo utilizar la recursividad para presentar una mejor solución a ciertos problemas.
  - Se empieza con el ejemplo del cálculo del factorial de cierto número. Dicho cálculo se realiza primeramente con el uso de la recursividad, en donde se utiliza un ciclo for que inicia a partir del valor pasado como argumento y va disminuyendo de uno en uno hasta llegar al uno. Cada que se repite el ciclo, se va multiplicando el número de índice de ese momento con la variable que nos dará como resultado el factorial. Es así como se obtiene la multiplicación de cada uno de los números.
    Este ciclo for puede ser reemplazado con una función recursiva, la cual multiplica el valor que fue pasado como argumento por una llamada a si misma, decrementando el valor pasado en uno, hasta que se llega al caso base en donde el valor valga uno y entonces se retorna el valor de 1. Esto se realiza con la finalidad de poder multiplicar todos los números y obtener el factorial deseado.
  - En el segundo ejemplo se presenta un programa que nos muestra gráficamente las pisadas o huellas que deja una tortuga, describiendo una forma de espiral. De igual manera, se muestran dos formas de realizarlo, una en donde se utiliza un ciclo for y otra en donde se utiliza la recursividad. Para el ciclo for, se le dice que realizará una acción 30 veces, la cual consiste en ir imprimiendo las huellas de la tortuga a lo largo de la pantalla, siguiendo una forma de tipo espiral.
    Para la recursividad, se realiza lo mismo, solamente que ahora se tiene la llamada a sí misma, en donde la cantidad de huellas que se pasa como argumento de disminuye en uno hasta que se llega al caso base en donde el número de huellas es igual a cero.
  - Por último, se presenta como ejemplo encontrar cierto término de la sucesión de Fibonacci. Al igual que en los casos anteriores, se muestra la manera recursiva y por iteración. Por iteración es tener los dos valores iniciales de la serie (0 y 1) y con un ciclo for, ir generando todos los demás términos, hasta llegar al término deseado.

En el caso de la recursividad, se plantean dos formas de realizarlo. Una en donde se utiliza puramente recursividad. Se llama a sí misma dos veces, una en donde el número pasado como argumento es disminuido en uno y otra en donde el número pasado como argumento es disminuido en dos. Ambos resultados se suman, repitiéndose el proceso hasta que se llega al caso base en donde número sea o igual a 1, entonces se retorna el valor de cero, o cuando se llega al caso base en donde el número sea igual a 2 o 3, entonces se retorna el valor de uno.

La segunda forma de realizarlo utiliza también una técnica denominada como memorización, en donde antes de realizar el proceso de recursividad descrito previamente, se verifica que el valor buscado no haya sido calculado previamente ya que, en caso de cumplirse esta condición, solamente se devuelve el valor almacenado en memoria. Esta memoria también sirve para establecer el caso base de la función recursiva, ya que desde un inicio se guardan los tres primeros valores de la serie de Fibonacci.

Al finalizar, se habla de algunas de las posibles desventajas de la recursividad, puesto que muchas veces resulta ser compleja de realizar y/o se tiene que la memoria y/o tiempo gastado es bastante alto.

Actividad 2: Para esta actividad se pidió describir y analizar diversas funciones de tipo recursivo, es decir, que se llaman a sí mismas.

1. En el ejercicio 1, se tiene una función denominada como funcion1, a la cual se le pasa como argumento una lista, la cual va a ser recorrida a partitr de su primer elemento con la finalidad de multiplicar todos los elementos que contiene. La función retorna el valor de esta multiplicación.

Es una función recursiva, puesto que se llama a sí misma, y cada que lo hace se le pasa como argumento la misma lista inicial pero excluyendo el primer elemento.

El caso base es cuando la sublista a la que se llega solamente tiene un elemento restante, o se encuentra vacía. En caso de estar vacía, se retorna como valor un número 1, y en caso de solamente tener un elemento restante, se retorna como valor este único elemento de la lista.

El caso recursivo es cuando, al no cumplirse los dos casos anteriores, la función retorna la multiplicación del primer valor en la lista con la llamada a la misma función, dando como resultado una recursividad.

In [2]:
```
def funcion1(L):
    if L==[]:
        return 1
    elif len(L)==1:
        return L[0]
    else:
        return L[0]*funcion1(L[1:])

lista1=[1,2,3,4,5,6,7,8]
lista2=funcion1(lista1)
print(lista2)
```

```
40320
```

2. En el ejercicio 2, se tiene una función denominada como funcion3, la cual tomo como argumentos una lista y un numero n, en donde dicho número n indicará hasta que posición de la lista se recorrerá. Se recorre la lista con la finalidad de sumar todos los elementos hasta la posición indicada. La función retorna el valor de esta suma.
Es una función recursiva, pues se llama a sí misma, y cada que lo hace se le pasa como argumento la misma lista inicial pero excluyendo el primer elemento, así como el número n, disminuido en 1.
El caso base es cuando se llega a que el número n sea igual a uno. En este caso se retorna el primer elemento de la sublista en la que se encuentra en ese momento.
El caso recursivo es cuando, al no cumplirse la condición anterior, la función retorna la suma del primer valor en la lista con la llamada a la misma función, dando como resultado una recursividad.

In [3]:
```
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(lista1,8)
print(lista2)
```

```
36
```

3. En el ejercicio 3, se tiene una función denominada como move, la cual toma como argumento un número n, y tres “posiciones”, en donde el número n indicará las veces que se realizará el movimiento. Esta función no retorna valor alguno, solamente se imprime en pantalla el «movimiento» de las distintas posiciones.
Es una función recursiva, pues se llama a sí misma en dos ocasiones, y en cada ocasión que lo hace se le pasa como argumento el mismo número n decrementado en uno, así como las mismas posiciones iniciales pero cambiadas de orden. En la primera llamada a sí misma, se cambia el orden de la posición y y z. En la segunda llamada a sí misma, se cambia el orden de la posición x y z. En ambas llamadas a sí misma, se decrementa en uno el valor de n.

El caso base es cuando se llega a que el número n sea igual a uno. En este caso, se imprime en pantalla el movimiento final de los valores x y y, en esa instancia de la llamada a la función.
El caso recursivo es cuando, al no cumplirse la condición anterior, la función se llama a sí misma, y una vez que se llega al caso base, cuando se lleva a cabo el retroceso recursivo, se imprime en pantalla el movimiento de los valores x y y, en esa instancia de la llamada a la función, para después volverse a llamar a sí misma.

```
In [4]: 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")
```

```
move A to C
move A to B
move C to B
move A to C
move B to A
move B to C
move A to C
move A to B
move C to B
move C to A
move B to A
move C to B
move A to C
move A to B
move C to B
move A to C
move B to A
move B to C
move A to C
```

4. En el ejercicio 4, se tiene una función denominada como mystery, la cual toma como argumento una cadena de texto. Esta función compara que la cadena se igual por ambos extremos, es decir, que tenga los mismos caracteres con su posición análoga en la cadena de texto. En caso de que se cumpla esta condición, se retorna un booleano true, en caso contrario, se retorna un booleano false.
Es una función recursiva, pues se llama a sí misma, y cada vez que lo hace se le pasa como argumento la misma lista inicial, pero excluyendo el primer y último carácter de esta.
El caso base es cuando se llega hasta una cadena de texto en donde solo exista un solo carácter o ningún carácter. En este caso, la función retorna un booleano true.

El caso recursivo es cuando, al no cumplirse la condición anterior, la función compara que el primer y último carácter de la cadena sean iguales, haciendo que se cumpla lo mismo para todos los demás caracteres restantes en la cadena, y es aquí cuando la función se llama a sí misma, para poder ir recorriendo toda la cadena de texto.

```
In [5]: def mystery(S):
            N=S.split()
            print(N)
            N=''.join(N)
            print(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("Ingrese una cadena \n")
        print(mystery(test))
```

```
Ingrese una cadena
do od
['do', 'od']
dood
['oo']
oo
[]

True
```

- Conclusiones

  En esta práctica se logró visualizar las diversas maneras de como se emplea la recursividad mediante el análisis de varios programas. También se mejoró el entendimiento de las distintas partes que componen a una función recursiva y la manera en como se distinguen estas diversas partes, las cuales fueron vistas de manera teórica en clase.

  De igual manera, se logró observar que las funciones recursivas son bastante útiles, pues permiten simplificar muchas líneas de código que se generan al utilizar diversas recursividades usando for, while, etc. Asimismo, muchas veces, cuando se combina la recursividad con otras técnicas, se logra una solución más eficaz, como es el caso de la combinación con la memorización, donde se guardan resultados previamente calculados.

  La recursividad es una herramienta bastante útil que permite simplificar en gran medida la resolución de problemas de distintas categorías, sin embargo, muchas veces resulta complicada de realizar, pues se necesita entender muy bien la lógica que sigue la recursividad, ya que en caso de no tenerla bien presenta, se podría caer en fallos como una recursividad interminable, una lógica circular, etc.

  El objetivo no se cumplió en su totalidad. Si bien se lograron analizar de manera correcta los distintos ejercicios propuestos y los ejemplos proporcionados en la guía, no se logró crear un código hecho por nosotros para resolver un problema utilizando recursividad.

La práctica permite darnos una idea bastante clara y entender de mejor manera las funciones recursivas, lo único que faltaría por añadir a la práctica sería un ejercicio en donde se pidiera resolver un problema sencillo haciendo uso de recursividad. De esta manera, se ahondaría de mejor manera en el tema de recursividad y se lograría una mejor visualización del concepto, ventajas, etc. De el uso de la recursividad para la solución de problemas.
Por lo demás, una práctica bastante útil e interesante que permite relacionar los conceptos vistos en clase.