**Practica 04 Soluciones Recursivas**

**Introducción**

Esta práctica tiene como fundamento la recursividad para dar soluciones a problemas en general, de una manera simplificada.

**4.1 Serie de Fibonacci**

La sucesión de Fibonacci (serie de Fibonacci) es la siguiente sucesión infinita de números naturales: 0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597…

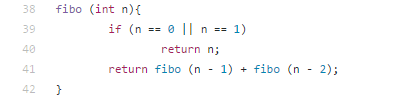
Esta sucesión fue descrita en Europa por Leonardo de Pisa, matemático italiano del siglo XIII también conocido como Fibonacci. Tiene numerosas aplicaciones en ciencias de la computación, matemática y teoría de juegos. También aparece en configuraciones biológicas, como por ejemplo en las ramas de los árboles, en la disposición de las hojas en el tallo, en las flores de alcachofas y girasoles, en las inflorescencias del brécol romanesco y en la configuración de las piñas de las coníferas.

El programa calcula el n-ésimo término de la sucesión de Fibonacci como f(n), con las condiciones iniciales f(n)=0 si n=0 y f(n)=1 si n=1, de otra manera la forma de calcular la suma de los dos números anteriores se define como f(n-1) +f(n-2) siempre y cuando n > 1.

**4.1.1 Planteamiento del problema**

Calcular el n-ésimo término de la sucesión de Fibonacci usando recursividad, considerando las condiciones iniciales.

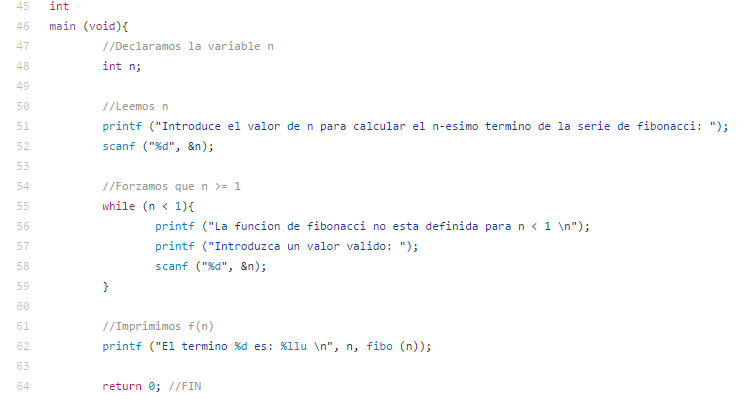
**4.1.2 Diseño y funcionamiento de la solución**



La función fibo recibe como parámetro único el valor que se quiere calcular de la sucesión de Fibonacci, respetando las condiciones iniciales, calculando la suma de los dos números anteriores.

**4.1.3 Implementación de la solución**

Teniendo la función fibo ya mencionada en el punto 4.1.2.



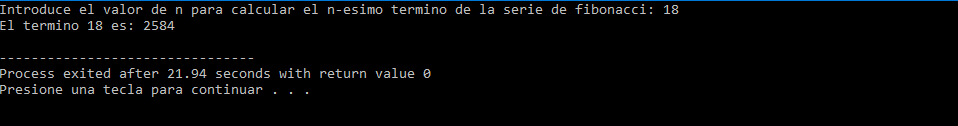
Se hace la declaración de una variable de tipo entero la cual va a ser el término que vayamos a calcular. (48)

Se le pide al usuario el n-ésimo termino a calcular mediante la entrada del teclado. (51-52)

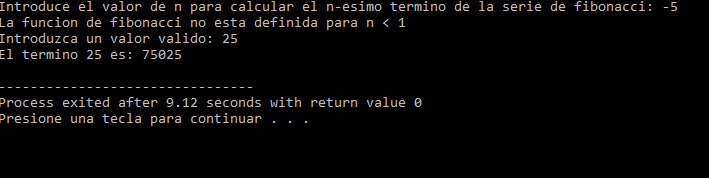
Pasando de las condiciones iniciales, se hace validación de que el termino n que introdujo el usuario sea n>1, de no ser así se le pide que introduzca un valor valido. (55-59)

Se muestra el termino n de la función mandando a llamar a la función fibo(n).

**4.1.4 Funcionamiento**



Ejecutando el programa e introduciendo un valor válido devuelve el termino introducido de la sucesión de Fibonacci.



De otra manera introduciendo un término no valido y vuelve a pedir el termino para mostrar el resultado.

**4.1.5 Errores detectados**

Debido a que la función es recursiva y entre más grande sea el término, más costo computacional tiene hacer los cálculos.



**4.1.6 Posibles mejoras**

Como la posible mejora más evidente seria implementar el programa de manera iterativa, ya que permitiría un mayor número para el cálculo, así como se reduce bastante el costo computacional que tendría al sacar el termino n-ésimo.

**4.2 Tribonacci**

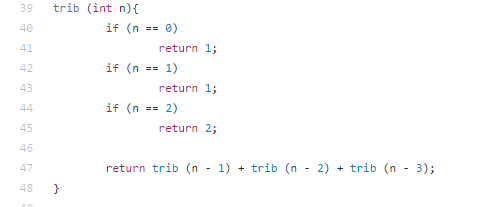
La sucesión de Tribonacci es una variante dela sucesión de fibonacci, pero sumando cada vez los tres anteriores, para lo que se empieza con 2 ceros: 0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, 274, 504, 927, 1.705, 3.136, 5.768, 10.609, 19.513, 35.890 ... Están relacionados con el polinomio –x³–x²–x+1 y.

El programa calcula el n-ésimo término de la sucesión de Tribonacci como f(n), con las condiciones iniciales t(n)=1 si n=0, t(n)=1 si n=1 y t(n)=2 si n=1, de otra manera la forma de calcular la suma de los tres números anteriores se define t(n - 1) + t(n - 2) + t(n - 3) siempre y cuando n > 2.

**4.2.1 Planteamiento del problema**

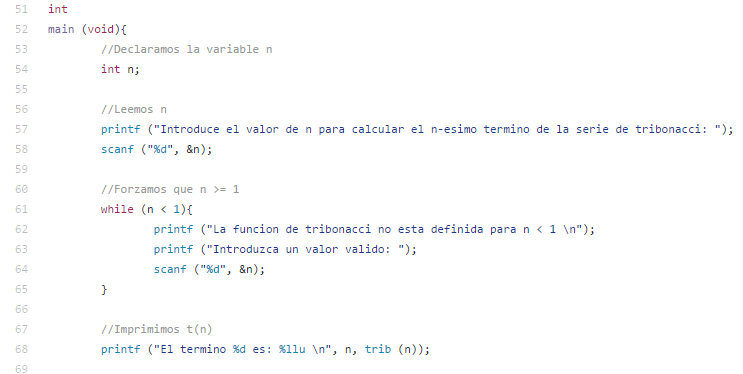
Calcular el n-ésimo término de la sucesión de Triibonacci usando recursividad, considerando las condiciones iniciales.

**4.1.2 Diseño y funcionamiento de la solución**



La función trib recibe como parámetro único el valor que se quiere calcular de la sucesión de Tribonacci, respetando las condiciones iniciales, calculando la suma de los tres números anteriores.

**4.1.3 Implementación de la solución**

Teniendo la función fibo ya mencionada 

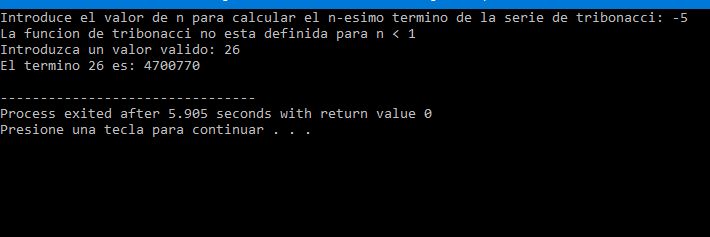
Se hace la declaración de una variable de tipo entero la cual va a ser el término que vayamos a calcular. (54)

Se le pide al usuario el n-ésimo termino a calcular mediante la entrada del teclado. (57-58)

Pasando de las condiciones iniciales, se hace validación de que el termino n que introdujo el usuario sea n>1, de no ser así se le pide que introduzca un valor valido. (55-59)

Se muestra el termino n de la función mandando a llamar a la función fibo(n).

**4.1.4 Funcionamiento**



Introduciendo un término no valido y vuelve a pedir el termino para mostrar el resultado. Luego introduciendo un valor válido devuelve el termino introducido de la sucesión de Tribonacci.

**4.1.5 Errores detectados**

Debido a que la función es recursiva y entre más grande sea el término, más costo computacional tiene hacer los cálculos.

**4.1.6 Posibles mejoras**

Como la posible mejora más evidente seria implementar el programa de manera iterativa, ya que permitiría un mayor número para el cálculo, así como se reduce bastante el costo computacional que tendría al sacar el termino n-ésimo.

**4.2 Torres de Hanói**

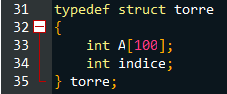
Las Torres de Hanói es un rompecabezas o juego matemático inventado en 1883 por el matemático francés Édouard Lucas. Este juego de mesa solitario consiste en un número de discos de radio creciente que se apilan insertándose en una de las tres estacas de un tablero. El objetivo del juego es crear la pila en otra de las estacas siguiendo ciertas reglas.

**4.3.1 Planteamiento del problema**

Resolver el problema de las torres de Hanói con n discos apilados de mayor a menor

**4.3.2 Diseño y funcionamiento de la solución**

Definimos una estructura torre, la cual contendrá los discos que se encuentran apilados en ella. Se hace el modelado de un disco con un entero n.



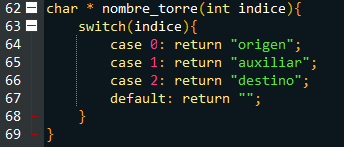
Dado un disco y una torre, añade el disco arriba de la torre, recibe un entero n el cual va a ser el número de discos y con el índice de la torre mayor a 1.



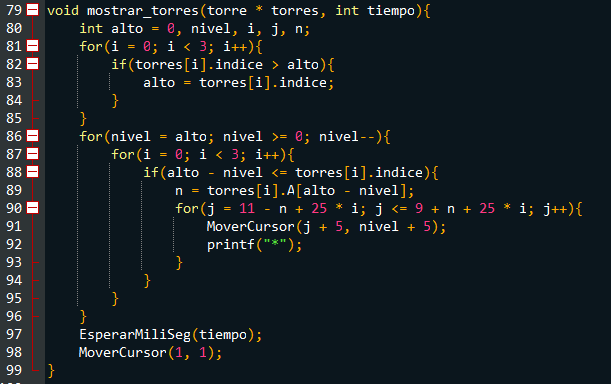
Dada una torre, remueve el disco que se encuentra hasta el tope que recibe un apuntador a torre T y devuelve un entero, el disco que se removió del tope de la torre, el índice de la torre debe ser mayor o igual a 0



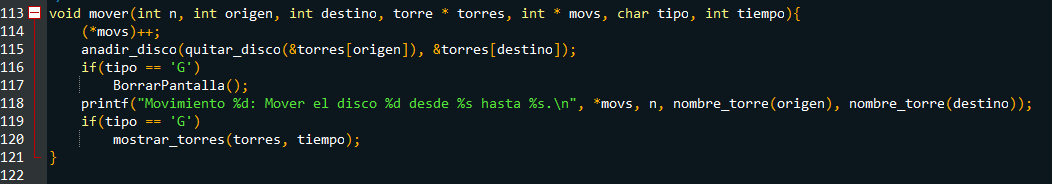
Dado un índice de torre, devuelve el nombre asociado a esta, que recibe un entero índice el cual es el índice de la torre, para que devuelva una cadena, indicando el nombre de la torre. Con el índice que debe estar entre 0 y 2, inclusive.



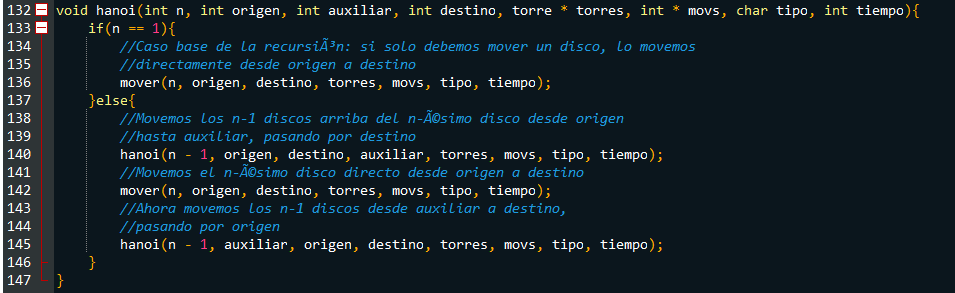
Dado un arreglo de torres y un tiempo, esta función imprime en pantalla de manera gráfica cómo se verán las torres de Hanói en un momento determinado, al leer el contenido de cada torre. Asumimos que el n-ésimo disco tiene un ancho en pantalla de 2n-1. Por cada estado, esperamos unos segundos y borramos la pantalla, recibe un arreglo de torres, entero que es tiempo y se da en milisegundos.



Dado un numero de disco con los índices de las torres (origen, destino), un arreglo de torres, un apuntador al número de movimientos, un char tipo y un entero tiempo la función mueve el n-ésimo disco desde origen hasta destino, incrementando en 1 el número de movimientos.



Esta es la función más importante ya que resuelve de manera recursiva el problema de las torres de Hanói dado el número de discos, los índices de las torres (origen, auxiliar, destino), un arreglo de torres, un apuntador a entero del número de movimientos, un char tipo y un tiempo que es en milisegundos. Si el char tipo es 'T', solo se mostrará la secuencia de movimientos requerida para resolver el problema, si es 'G', además se animará en pantalla el estado de cada movimiento.



**4.3.3 Implementación de la solución**

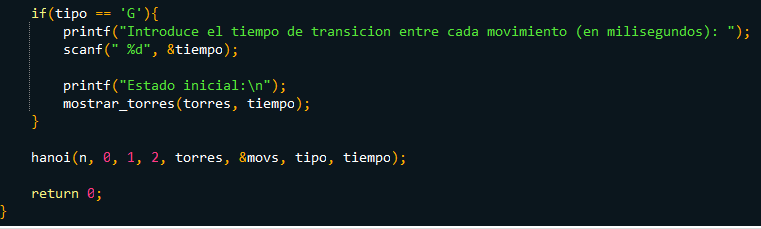
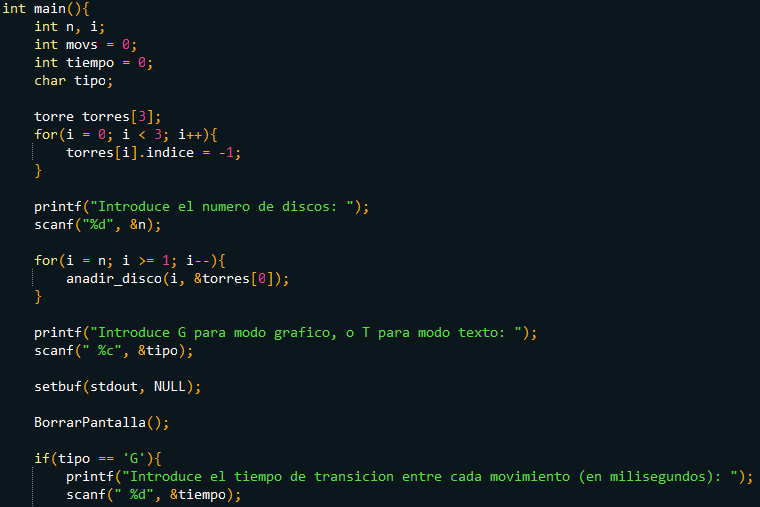
En la implementación se declaran las variables que se ocuparán siendo n los discos, los movimientos, el tiempo en milisegundos y el tipo.

Se inicializa un arreglo de torres y se le pide al usuario que introduzca el número de discos y leemos la entrada.

Se agregan los n discos a la torre de origen donde empezaran los movimientos para poder visualizar la animación. Se le pregunta al usuario qué modo de solución quiere, grafico o solo el texto.

Se hace la limpieza de pantalla y se valida si el usuario quiere el modo grafico o solo texto, si pide el método grafico se le pide un tiempo que será en milisegundos para pasar las transiciones de la animación de las torres y se manda a llamar la función mostrar torres para hacer la animación.

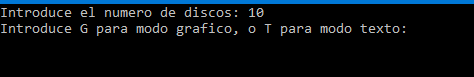
Y para la resolución del problema recursivo se manda a llamar a la función hanoi que prácticamente es la que hace la solución.



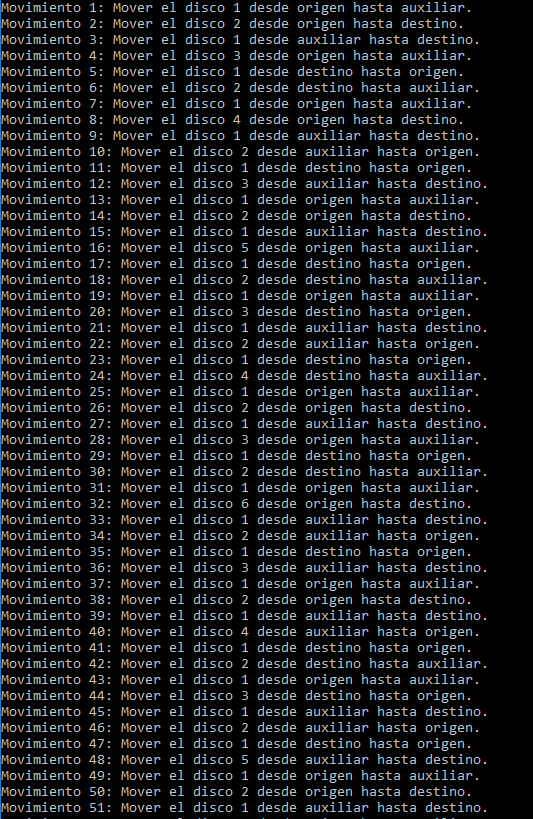
**4.3.4 Funcionamiento**

Al ejecutar el programa lo primero que hace es pedir al usuario que introduzca el número de discos.

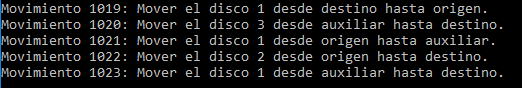
Posteriormente le pide si quiere las soluciones por método grafico o por texto mediante las letras G y T respectivamente



En caso de elegir el modo de texto se despliegan las soluciones de como mover cada disco y a que asta.



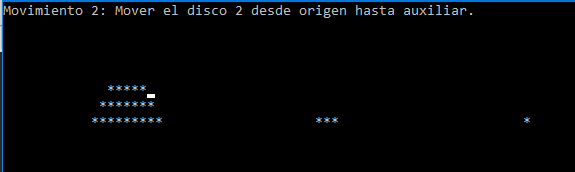
En el caso de 10 discos nos muestra los 1023 movimientos que tenemos que hacer.

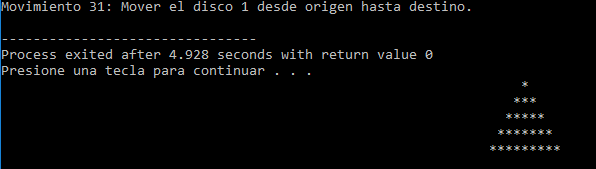


Si el usuario elige el modo grafico en este caso con 5 discos después de introducir el modo G para mostrar los movimientos, el programa le pide al usuario una ultima entrada la cual va a ser el tiempo de transición entre cada movimiento de los discos en milisegundos.



Después de ser introducido el tiempo ejecuta los movimientos hasta llegar a la solución.





**4.3.5 Errores detectados**

En la forma de soluciones gráficas solo se pueden visualizar de forma correcta a lo más 12 discos, al introducir a n como más de 12 el programa no muestra de manera correcta los discos apilados de mayor a menor, ya que la resolución de la consola con respecto a la pantalla no alcanza a imprimir tantos caracteres.



**4.1.6 Posibles mejoras**

Debido a que el programa muestra visualmente a los más 12 discos apilados de mayor a menor para la forma gráfica, la solución más adecuada sería buscar e implementar la forma de que se puedan ver más discos.

**4.3 Conclusiones**

4.3.1 Alan

4.3.2 Ángel

4.3.3 Luis

En conclusión, la recursividad nos puede ayudar bastante para la resolución de problemas de una manera que se pueda ver sencilla, pero esto no quiere decir que hacer programas de manera recursiva lo sea, y que requiere de cierta complejidad para hacer el análisis, plantear y dar solución a la problemática que queramos llevar a cabo. Hay que tener en cuenta que la recursividad va a hacer las cosas de manera simple en ciertos casos, y en otros no sería tan conveniente usar una función recursiva, ya que como lo vimos en la serie de Fibonacci después de un cierto número que queramos calcular va a ser más tardado y por lo tanto tendrá un alto costo computacional, hacer las cosas recursivas nos va a ayudar mucho si sabemos cómo y cuándo podemos hacerlo.

**4.4 Anexos**

\*Meter todo el código :v\*