INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Estructura de Datos

## 191Xdfef@1CM15S



Practica 4

Prof.: Franco Martínez Edgardo Adrián

Integrantes:

Mendoza Parra Sergio.

Salcedo Barrón Ruben Osmair

Tejeda Martínez José Miguel.

MEXICO, D.F. a 9 de Enero del 2016

**Introducción:**

En este programa se implementa el algoritmo de la recursividad con el problema de las torres de “HANOI” que dependiendo de cuantos discos ingrese el usuario el programa da el numero de pasos a seguir y a su vez se implementa de forma recursiva la sucesión de Fibonacci y de tribonacci que dependiendo de la recursividad que nos del número más grande que se ingrese.

**Planteamiento del problema:**

Algoritmo de Recursividad: Para este problema lo que hicimos fue entender primero el algoritmo de recursividad con algunos ejemplos para poder resolver el problema y a su vez jugar en algunas páginas algunos juegos de las torres de Hanói para ver cómo se tenían que mover los discos y ver esa recursividad en las torres.

Nos dimos cuenta de la recursividad porque cuando son números pares los discos, el primer movimiento es en la torre 2 y si son números impares el primer movimiento es en la torre 1 y también si son muchas torres se deben de contar cuantas torres se tienen para ver si son pares e impares.

En la parte de Fibonacci y tribonacci se hizo lo mismo que el de las torres de Hanói lo que hicimos fue que nos basamos principalmente en el algoritmo de Fibonacci y tribonacci normal con sus valores iniciales ya que en la parte del Fibonacci f[0]=0 y f[1]=1 lo que se hizo fue exactamente lo mismo pero poniéndolo en una función de tal manera que al mandarla a llamar en ella misma se diera lo mismo que como el programa clásico y de la parte de tribonacci se hizo lo mismo que el de Fibonacci.

**Diseño y funcionamiento de la solución:**

**“TORRES DE HANOI”**

**Recursividad**

En base al programa y en base al planteamiento del problema, tomamos en cuenta la parte de la recursividad ya que lo que hicimos fue que pusimos o entendimos la parte de recursividad de la siguiente manera:

Número de discos impares: Ejemplo:

Los movimientos son los siguientes:

1. Primer disco a torre 3
2. Segundo disco a torre 2
3. Primer disco a torre 2
4. Tercer disco a torre 3
5. Primer disco a torre 1
6. Segundo disco a torre 3
7. Primer disco a torre 3

Como podemos ver si tenemos un número de disco mayor e impar, por ejemplo el numero de discos 27 el primer movimiento siempre será a la torre 3 siempre y después teniendo la torre auxiliar que es la torre 2, si tenemos de 2 a más discos tenemos que contar los discos de tal manera que la torre 2 queda como a torre 1 y si el numero de discos es impar el primer movimiento es a torre 1 y el otro a torre 3 y así sucesivamente

Número de discos pares: Ejemplo:

Los movimientos son los siguientes:

1. Primer disco a torre 2
2. Segundo disco a torre 3
3. Primer disco a torre 3

En este caso sería la misma explicación sólo que como los discos son números pares nuestro auxiliar seria la torre 2 siempre y así sucesivamente como la anterior explicación.

**Fibonacci**

**Recursividad**

En base al programa y en base al planteamiento del problema, tomamos en cuenta la parte de la recursividad ya que como lo mencionado antes nos basamos en el programa clásico de Fibonacci sin recursividad en este caso lo que hicimos fue siguiente:

La formula nos dice que f[n]=f [n-1]+f [n-2] de tal manera que f [0]=0 y f [1]=1 por lo tanto si nosotros ponemos lo mismo en una función que tenga como parámetro una “n”:

Podemos hacerlo siguiente matemáticamente:

Dom= n ≥ 0; n=n V n= (n-1) + (n-2)

**Tribonacci**

**Recursividad**

En base al programa y en base al planteamiento del problema, tomamos en cuenta la parte de la recursividad que de igual manera como se explicó lo de Fibonacci es exactamente lo mismo de hecho solo en la formula se agrega un tercer digito lo cual sería:

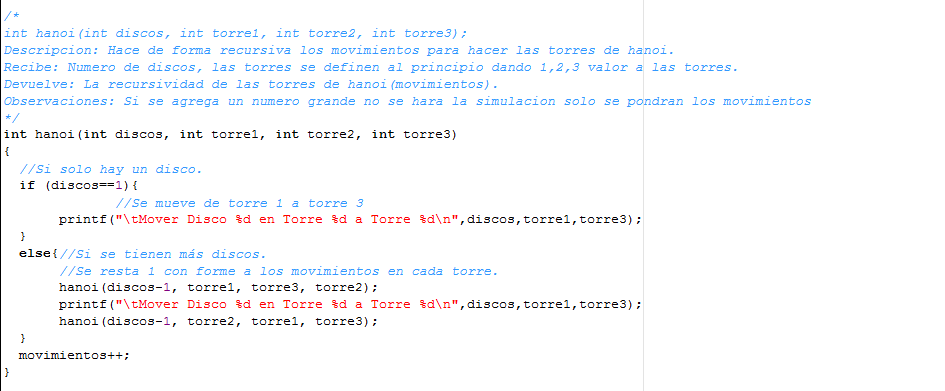
F[n]=f [n-1] + f [n-2] + f [n-3]

Dom= n ≥ 0; n=n V n= (n-1) + (n-2) + (n-3)

**Implementación de la solución en el código:**

**TORRES DE HANOI**

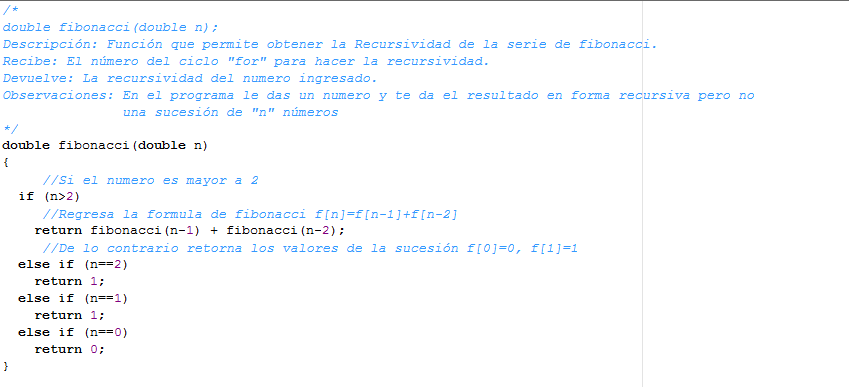
**Recursividad**



En este caso sería lo mismo que se explico como en el primer caso si hay un disco ese disco se va a la torre 3, pero si hay más que sean pares los discos se van a la torre 2 por eso se cambió en el “else” la torre 3 por la torre 2 porque ahora la torre 3 va a hacer nuestro auxiliar y con base a eso pasar los discos de tal manera que se resuelva la recursividad.

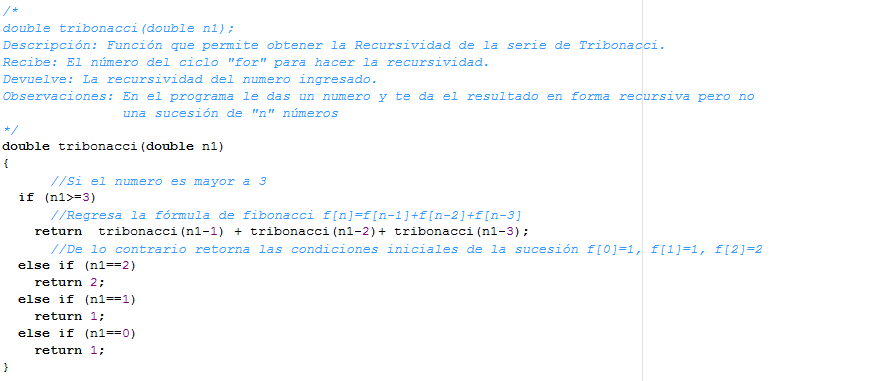
**FIBONACCI**

**Recursividad**



Aquí lo que podemos observar conforme a lo explicado tenemos que hacer una función que tenga un parámetro n el cual va a hacer el que le vamos a dar un valor y si el valor es 0, 1 ,2 cae en las condiciones las cuales van a empezar a dar la sucesión.

**TRIBONACCI**

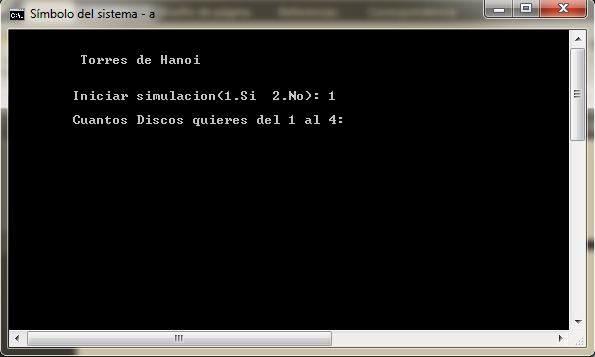
****

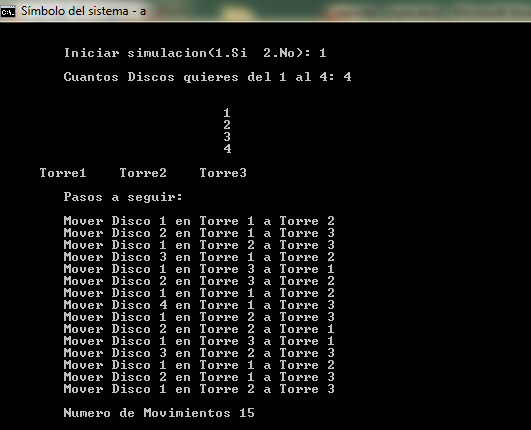
El funcionamiento es el mismo conforme al de Fibonacci de hecho es igual solo que las condiciones iniciales cambian ya que en tribonacci se agrega un “n-3” en la fórmula.

**Funcionamiento**

Funcionamiento del programa: **TORRES DE HANOI**

Aquí se puede observar que primero se pregunta si se quiere empezar el programa y después pregunta cuantos discos quieres, solo que hay un detalle la simulación solo se puede de 1 a 4 porque si se ingresa más discos solo se va a poder ver los pasos a seguir.

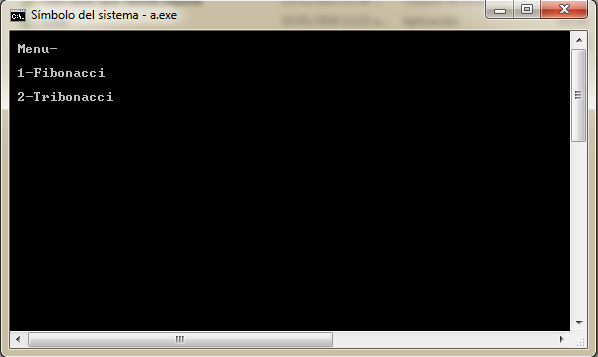


Después de haber ingresado el numero de discos se puede ver la simulación (aquí no verá porque es en imagen).

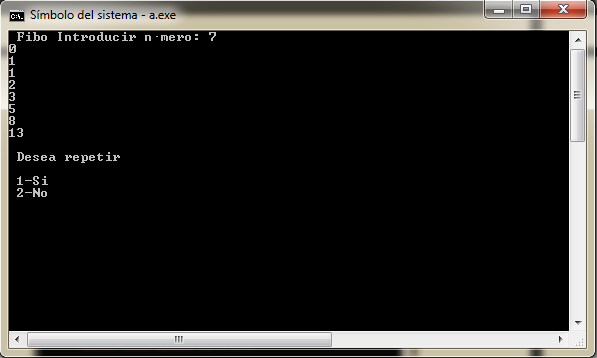
**“FIBONACCI”**

Funcionamiento del programa:

Aquí es la parte del menú, el usuario tiene que ingresar que opción quiere.



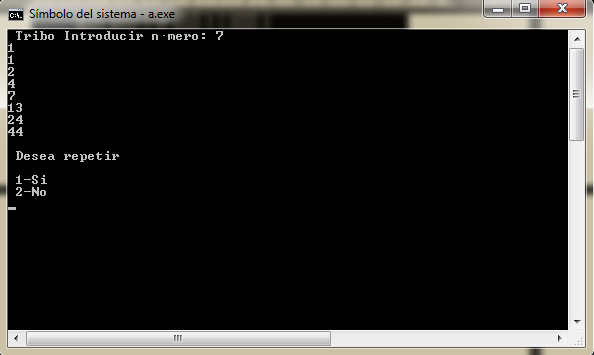
Una vez ingresada la opción que quiere se pide un número para que calcule el número que dio el usuario en Fibonacci. Por ejemplo:



**“TRIBONACCI”**

Funcionamiento del programa:

Una vez ingresada la opción de tribonacci ingresamos un número cualquiera y se calcula el número que dio el usuario en tribonacci.

****

**Errores detectados**

Los errores detectados los siguientes:

Torres de Hanói:

1. No se pudo hacer las simulaciones de torres de Hanói para n discos de forma recursiva.

Fibonacci:

1. No se pudo ingresar números muy grandes porque se atoraba el programa.

Tribonacci:

1. Igual lo mismo no se pudo ingresar un número muy grande porque se atoraba el programa.

**Posibles mejoras**

Se puede disminuir el código:

Torres de Hanói.

1. Se puede hacer mejoras de forma recursiva la simulación de cuando se resuelve las torres de Hanói.

Fibonacci y Tribonacci:

1. Hay una librería que utiliza números grandes pero al momento de ingresarla el programa no la reconoce pero podría ser una posible mejora.

**Anexos**

(Código Fuente):

**Torres de Hanoi:**

/\*

TORRES DE HANOI

AUTOR: Mendoza Parra Sergio, Salcedo Barron Ruben Osmair, Tejeda Martinez Jose Miguel (C) Diciembre 2015

VERSIÓN: 1.0

DESCRIPCIÓN: Con la implementación de "Recursividad"

realizar la simulacion de las torres de hanoi

el cual se tendra que ver de forma gráfica,

los movimientos que realiza cada disco.

OBSERVACIONES: El programa permite agregar cualquier número

de discos pero, del 1 al 4 hay simulación y

mayor a 4 solo estan los pasos a seguir y el

numero de movimientos que se deben realizar.

EL CODIGO QUE IMPLEMENTA LA SIMULACION ES: Libreria.c

COMPILACION: gcc Hanoi.c Libreria.c -o "Nombre del ejecutable"

EJECUCION: ./"Nombre del ejecutable" (Linux)

"Nombre del ejecutable".exe (Windows)

\*/

//Librerias

#include <stdio.h>

#include <stdlib.h>

#include "Libreria.h" //Libreria que nos permite usar Gotoxy(MoverCursor) para la simulacion

#define tiempo 2 //Tiempo que se espera en cada paso de la simulación

//Funciones

int hanoi(int discos, int torre1, int torre2, int torre3);

int borrar(int x,int y);

int graficos(int discos);

//Variable Global (Contador de movimientos)

int movimientos=0;

int main()

{

menu();

}

int Borrar(int x,int y)

{

Gotoxy(x,y);

printf(" ");

}

/\*

int graficos(int discos);

Descripcion: Hace la simulacion de las torres de hanoi menores a 5 y mayor a 0.

Recibe: Numero de discos para realizar las simulacion con sus respectivos casos.

Devuelve: La simulación con respectivos discos.

Observaciones: Si se agrega un numero grande no se hara la simulacion solo se pondran los movimientos

\*/

int graficos(int discos)

{

int x;

//Posiciones para simulación

int y=13;

int x1=5;

if(discos==1){

Gotoxy(x1,y);

for(x=1;x<=discos;x++)

{

printf("%d\n",x);

}

//Se espera 2 segundos en cada movimiento de la simulacion

sleep(tiempo);

Borrar(x1,y);

Gotoxy(x1+23,y);

printf("%d",x-1);

sleep(tiempo);

printf("\n\n\n\n\n");

}

if(discos==2){

Gotoxy(x1,y-1);

for(x=1;x<=discos;x++)

{

printf("\t%d\n",x);

}

sleep(tiempo);

Borrar(x1,y-1);

Gotoxy(x1+14,y);

printf("%d",x-2);

sleep(tiempo);

Borrar(x1,y);

Gotoxy(x1+23,y);

printf("%d",x-1);

sleep(tiempo);

Borrar(x1+14,y);

Gotoxy(x1+23,y-1);

printf("%d",x-2);

sleep(tiempo);

printf("\n\n\n\n\n");

}

if(discos==3){

Gotoxy(x1,y-2);

for(x=1;x<=discos;x++)

{

printf("\t%d\n",x);

}

sleep(tiempo);

Borrar(x1,y-2);

Gotoxy(x1+23,y);

printf("%d",x-3);

sleep(tiempo);

Borrar(x1,y-1);

Gotoxy(x1+14,y);

printf("%d",x-2);

sleep(tiempo);

Borrar(x1+23,y);

Gotoxy(x1+14,y-1);

printf("%d",x-3);

sleep(tiempo);

Borrar(x1,y);

Gotoxy(x1+23,y);

printf("%d",x-1);

sleep(tiempo);

Borrar(x1+14,y-1);

Gotoxy(x1+3,y);

printf("%d",x-3);

sleep(tiempo);

Borrar(x1+14,y);

Gotoxy(x1+23,y-1);

printf("%d",x-2);

sleep(tiempo);

Borrar(x1,y);

Gotoxy(x1+23,y-2);

printf("%d",x-3);

sleep(tiempo);

printf("\n\n\n\n\n");

}

if(discos==4){

Gotoxy(x1,y-3);

for(x=1;x<=discos;x++)

{

printf("\t%d\n",x);

}

sleep(tiempo);

Borrar(x1,y-3);

Gotoxy(x1+14,y);

printf("%d",x-4);

sleep(tiempo);

Borrar(x1,y-2);

Gotoxy(x1+23,y);

printf("%d",x-3);

sleep(tiempo);

Borrar(x1+14,y);

Gotoxy(x1+23,y-1);

printf("%d",x-4);

sleep(tiempo);

Borrar(x1,y-1);

Gotoxy(x1+14,y);

printf("%d",x-2);

sleep(tiempo);

Borrar(x1+23,y-1);

Gotoxy(x1+3,y-1);

printf("%d",x-4);

sleep(tiempo);

Borrar(x1+23,y);

Gotoxy(x1+14,y-1);

printf("%d",x-3);

sleep(tiempo);

Borrar(x1,y-1);

Gotoxy(x1+14,y-2);

printf("%d",x-4);

sleep(tiempo);

Borrar(x1,y);

Gotoxy(x1+23,y);

printf("%d",x-1);

sleep(tiempo);

Borrar(x1+14,y-2);

Gotoxy(x1+23,y-1);

printf("%d",x-4);

sleep(tiempo);

Borrar(x1+14,y-1);

Gotoxy(x1+4,y);

printf("%d",x-3);

sleep(tiempo);

Borrar(x1+23,y-1);

Gotoxy(x1+4,y-1);

printf("%d",x-4);

sleep(tiempo);

Borrar(x1+14,y);

Gotoxy(x1+23,y-1);

printf("%d",x-2);

sleep(tiempo);

Borrar(x1,y-1);

Gotoxy(x1+14,y);

printf("%d",x-4);

sleep(tiempo);

Borrar(x1,y);

Gotoxy(x1+23,y-2);

printf("%d",x-3);

sleep(tiempo);

Borrar(x1+14,y);

Gotoxy(x1+23,y-3);

printf("%d",x-4);

sleep(tiempo);

printf("\n\n\n\n\n");

}

}

/\*

int hanoi(int discos, int torre1, int torre2, int torre3);

Descripcion: Hace de forma recursiva los movimientos para hacer las torres de hanoi.

Recibe: Numero de discos, las torres se definen al principio dando 1,2,3 valor a las torres.

Devuelve: La recursividad de las torres de hanoi(movimientos).

Observaciones: Si se agrega un numero grande no se hara la simulacion solo se pondran los movimientos

\*/

int hanoi(int discos, int torre1, int torre2, int torre3)

{

//Si solo hay un disco.

if (discos==1){

//Se mueve de torre 1 a torre 3

printf("\tMover Disco %d en Torre %d a Torre %d\n",discos,torre1,torre3);

}

else{//Si se tienen más discos.

//Se resta 1 con forme a los movimientos en cada torre.

hanoi(discos-1, torre1, torre3, torre2);

printf("\tMover Disco %d en Torre %d a Torre %d\n",discos,torre1,torre3);

hanoi(discos-1, torre2, torre1, torre3);

}

movimientos++;

}

int menu()

{

int discos;

int opcion;

int x=5,y=15,i;

system("cls");

printf ("\n\n\t Torres de Hanoi \n ");

printf("\n\n\tIniciar simulacion(1.Si 2.No): ");

scanf ("%d", &opcion);

if(opcion==1){

printf("\n\tCuantos Discos quieres del 1 al 4: ");

scanf("%d",&discos);

printf("\n\n\n\n\n");

//Posiciones de las torres para simulación.

Gotoxy(x,y);

printf("Torre1");

Gotoxy(x+10,y);

printf("Torre2");

Gotoxy(x+20,y);

printf("Torre3");

graficos(discos);

printf("\n\n");

printf("\tPasos a seguir:\n\n");

hanoi(discos, 1, 2, 3);

printf("\n\tNumero de Movimientos %d \n\n\n",movimientos);

system("pause");

main();

}

if(opcion>=2 || opcion<0){

exit(1);

}

}

**Fibonacci y Tribonacci:**

/\*

FIBONACCI Y TRIBONACCI

AUTOR: Mendoza Parra Sergio, Salcedo Barron Ruben Osmair, Tejeda Martinez Jose Miguel (C) Diciembre 2015

VERSIÓN: 1.0

DESCRIPCIÓN: Con la implementación de "Recursividad"

realizar de manera recursiva el numero máximo

en el que llega la serie de fibonacci y

Tribonacci.

OBSERVACIONES: El programa puede tener problemas dependiendo

del número en el que llegue las series por lo que

depende del programa hasta donde llegue.

COMPILACIÓN: gcc Menu.c -o "Nombre del ejecutable"

EJECUCION: ./"Nombre del ejecutable" (Linux)

"Nombre del ejecutable".exe (Windows)

\*/

//Librerias

#include <stdio.h>

#include <stdlib.h>

/\*

double fibonacci(double n);

Descripción: Función que permite obtener la Recursividad de la serie de fibonacci.

Recibe: El número del ciclo "for" para hacer la recursividad.

Devuelve: La recursividad del numero ingresado.

Observaciones: En el programa le das un numero y te da el resultado en forma recursiva pero no una sucesión de "n" números

\*/

double fibonacci(double n)

{

//Si el numero es mayor a 2

if (n>2)

//Regresa la formula de fibonacci f[n]=f[n-1]+f[n-2]

return fibonacci(n-1) + fibonacci(n-2);

//De lo contrario retorna los valores de la sucesión f[0]=0, f[1]=1

else if (n==2)

return 1;

else if (n==1)

return 1;

else if (n==0)

return 0;

}

/\*

double tribonacci(double n1);

Descripción: Función que permite obtener la Recursividad de la serie de Tribonacci.

Recibe: El número del ciclo "for" para hacer la recursividad.

Devuelve: La recursividad del numero ingresado.

Observaciones: En el programa le das un numero y te da el resultado en forma recursiva pero no una sucesión de "n" números

\*/

double tribonacci(double n1)

{

//Si el numero es mayor a 3

if (n1>=3)

//Regresa la fórmula de fibonacci f[n]=f[n-1]+f[n-2]+f[n-3]

return tribonacci(n1-1) + tribonacci(n1-2)+ tribonacci(n1-3);

//De lo contrario retorna las condiciones iniciales de la sucesión f[0]=1, f[1]=1, f[2]=2

else if (n1==2)

return 2;

else if (n1==1)

return 1;

else if (n1==0)

return 1;

}

int main()

{

int op,r;

double num,x;

double num1,x1;

do

{

system("cls");

printf ("\t\n Menu- \n\n 1-Fibonacci \n\n 2-Tribonacci \n ");

scanf ("\n %d", &op);

system("cls");

switch (op){

case 1:

printf (" Fibo Introducir número: ");

scanf ("%lf", &x);

//Se pasa por valor el numero a la funcion fibonacci.

for (num=0; num<=x; num++)

{

printf("%.lf\n",fibonacci(num));

}

break;

case 2:

printf (" Tribo Introducir número: ");

scanf ("%lf", &x1);

//Se pasa por valor el numero a la funcion tribonacci.

for (num1=0; num1<=x1; num1++)

{

printf("%.lf\n",tribonacci(num1));

}

break;

default: exit(1);

break;

}

printf ("\t\n Desea repetir \n\n 1-Si \n 2-No\n");

scanf ("\n %d", &r);

}while(r!=2);

return 0;

}

**Conclusiones**

**Tejeda Martínez José Miguel**: La recursividad es un concepto esencial en el desarrollo de código estructurado, ya que hay diferentes programas que pueden ser simplificados de tal modo que sus funciones estén definidas en términos de sí mismas. Como pudimos observar, para las secuencias de Fibonacci, así como para cualquier programa recursivo es necesario tener nuestros “casos base”, y a partir de ellos realizar las posteriores llamadas a la misma función.

El resultado de aplicar la recursividad resulta en código más simplificado y de mayor entendimiento, aunque las fallas que se podrían generar son en el manejo de miles de llamadas recursivas que lleguen al límite de la memoria destinada para su ejecución.

**Mendoza Parra Sergio**: En esta práctica se vio la parte de la recursividad ya que es un método que se utiliza para resolver problemas que son consecutivos como lo fue hanoi, fibonacci y tribonacci de tal manera que pudimos observar o entender la parte en la que se podía hacer la recursividad mandando a llamar la función con forme a la ecuación de cada problema y obviamente teniendo sus condiciones iniciales del problema.

Por otro lado ésta práctica se entendió ya que en mi punto de vista fue una práctica en la que tuvimos algunos problemas en la parte de fibonacci y tribonacci ya que al momento de hacer o calcular un numero grande se tardaba demasiado en hacer la sucesión y de la parte de hanoi solo tuvimos problemas en el numero de discos que se tienen.

**Salcedo Barrón Ruben**: Como vimos en el desarrollo de la práctica tuvimos que usar recursividad, así que fue buena opción para ver las diferentes maneras de pensar bien cómo implementar la función expresada en sí misma, además de que nos apoyamos de internet para entender mejor este concepto y una vez entendiendo lo que debía hacer la función fue sencillo implementarla en Fibonacci, tribonacci y las torres de hanoi.

En lo particular se me hizo más fácil entender el programa de Fibonacci y de ahí pudimos realizar el de tribonacci sin embargo el de torres de hanoi se me dificulto un poco más.

# BIBLIOGRAFÍA

#### [1] Edgardo Adrián Franco Martínez, Practica 04: [Practica 04: "Soluciones Recursivas"](http://eafranco.com/docencia/estructurasdedatos/files/practicas/Practica04.pdf) Recursos Adicionales:[Animación Hanoi](http://eafranco.com/docencia/estructurasdedatos/files/practicas/04/Hanoi.htm), [Juega Hanoi](http://britton.disted.camosun.bc.ca/hanoi.swf) , Disponible en:

<http://eafranco.com/docencia/estructurasdedatos/files/practicas/Practica04.pdf>