

Universidad Tecnológica De la Mixteca



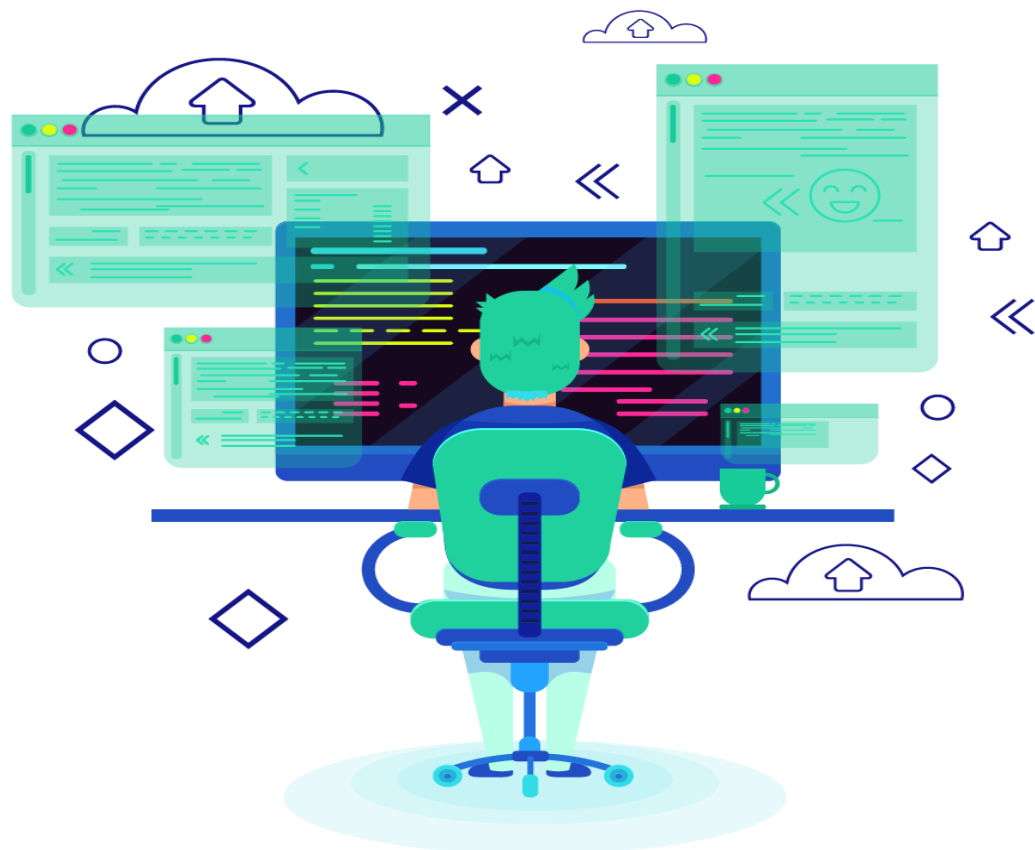
## Programación Estructurada

Alumno: Leonel Santiago Rosas

Grupo: 102-A

Catedrático: Dr. Christian Eduardo Millán  
Hernandez

Ejercicios 5.0 Del Libro de Deitel.



## Contents

Ejercicios 5.23: .....	2
Algoritmo: .....	2
Codificación: .....	2
Prueba 5.23: .....	3
.....	3
Ejercicio 5.27: .....	4
Algoritmo: .....	4
Prueba: .....	6
Ejercicio 5.35 .....	7
Algoritmo .....	7
Código .....	7
Prueba: .....	8
Ejercicio 5.38 .....	9
Algoritmo: .....	9
Prueba: .....	11
Ejercicios 5.39 .....	11
Código .....	12
Prueba: .....	13
Conclusiones: .....	13

### Ejercicios 5.23:

Escriba una función que obtenga el tiempo como tres argumentos enteros ( Para horas, minutos y segundos) y regrese el numero de segundos desde la ultima vez que el reloj “llego a las 12”. Utilice esta función para calcular la cantidad de tiempo en segundos entres 2 horas, cuando ambas estén dentro de un ciclo de 12 horas del reloj

#### Algoritmo:

1. Inicio
2. Insertar ambos tiempos
3. Convertir los tiempos a segundos multiplicando las horas por 3600 y los minutos por 60
4. Restar ambos
5. Imprimir el resultado
6. Salida

#### Codificación:

```
#include<stdio.h>
#include<stdlib.h>

int h,m,s;
int hh,mm,ss;
int tiempo1();
int main(){
    printf("\n Introduce el tiempo: Hrs, Min, seg: (ctrl+c)para finalizar\n");
    scanf("%d%d%d", &h, &m, &s);
    printf("\n Introduce el tiempo: Hrs, Min, seg: \n");
    scanf("%d%d%d", &hh, &mm, &ss);

    if (h<=12 && m<=60 && s<=60 && hh<=12 && mm<=60 && ss<=60 || h!=-1 )
    {
        h*=3600;
        m*=60;
        hh*=3600;
        mm*=60;
        tiempo1();
    }else
    {
        printf("\n Por favor ingresa un tiempo valido");
    }
}
```

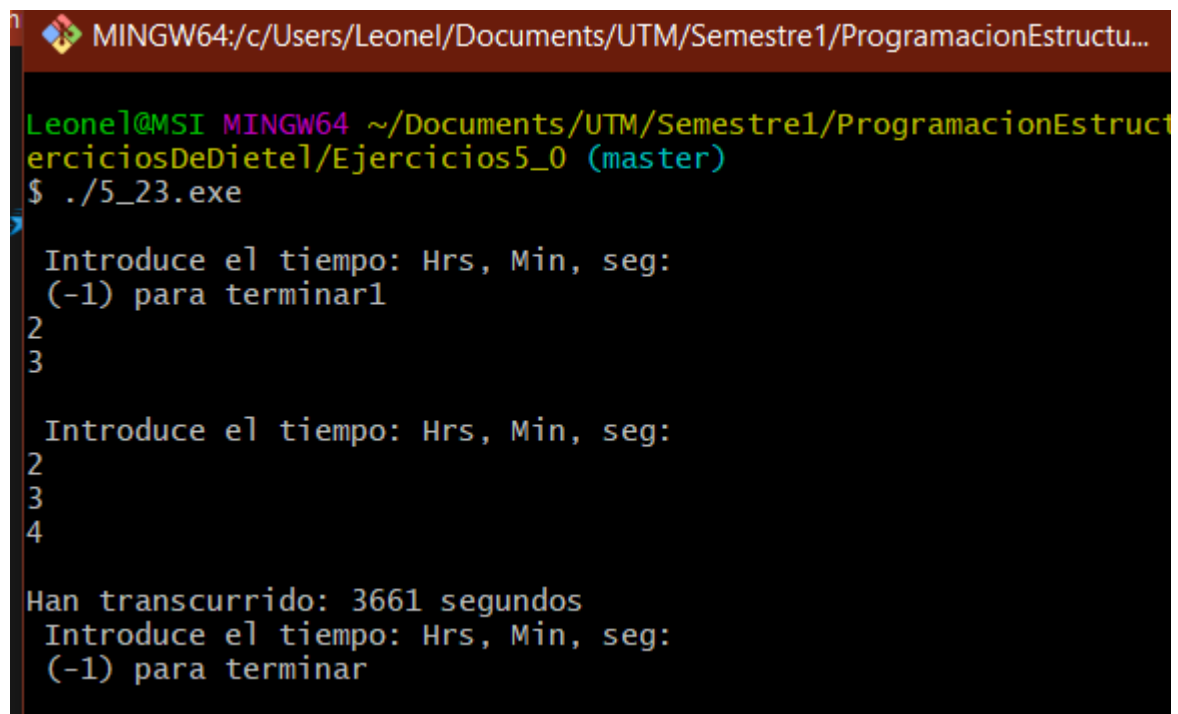
```

}

int tiempo1(){
    int h1;
    h1=(h+m+s)-(hh+mm+ss);
    if (h1>=0){
        printf("\n Han transcurrido: %d segundos", h1);
    }else{
        printf("\nHan transcurrido: %d segundos", h1*-1);
    }
    main();
}

```

Prueba 5.23:



```

MINGW64:/c/Users/Leonel/Documents/UTM/Semestre1/ProgramacionEstructu...
Leonel@MSI MINGW64 ~/Documents/UTM/Semestre1/ProgramacionEstructu...
EjerciciosDeDietel/Ejercicios5_0 (master)
$ ./5_23.exe

Introduce el tiempo: Hrs, Min, seg:
(-1) para terminar
2
3

Introduce el tiempo: Hrs, Min, seg:
2
3
4

Han transcurrido: 3661 segundos
Introduce el tiempo: Hrs, Min, seg:
(-1) para terminar

```

## Ejercicio 5.27:

**5.27** Se dice que un entero es *primo* si es divisible sólo entre 1 y sí mismo. Por ejemplo, 2, 3, 5, y 7 son primos, pero 4, 6, 8 y 9 no lo son.

- Escriba una función que determine si un número es primo.
- Utilice esta función en un programa que determine e imprima todos los números primos entre 1 y 10,000. ¿Cuántos de estos 10,000 números tendrá que probar verdaderamente antes de estar seguro de que se han encontrado todos los números primos?
- Inicialmente pudiera pensar que  $n/2$  es el límite superior para el cual debe usted probar para ver si un número es primo, pero sólo necesita llegar hasta la raíz cuadrada de  $n$ . ¿Por qué? Vuelva a escribir el programa, y ejecútelo de ambas formas. Estime la mejora en rendimiento.

## Algoritmo:

- Inicio
- Ingresar el numero
- Si es divisible entre si y entre 3 da 0 es primo
- Imprimir si el numero es primo

Codigo:

```
#include<stdlib.h>
#include<stdio.h>
int op;
void primo(), mil(), metodoraiz();
int main(){

    getchar();
    system("cls");
    printf("\n Opciones \n 1) Si 'n' es primo\n 2) Saber los numero primos del 1
al 10mil\n 3)1 a 10 mil segundo metodo\n 4)salir\n  ingresa una opcion: ");
    scanf("%d", &op);
    do
    {
        switch (op)
        {
            case 1: primo(); break;
            case 2: mil(); break;
            case 4: op=4; break;
        }
    } while (op!=4);
    return 0;
}

void primo(){
    int n, di=2 , pri=0;
    printf("\n Ingrese un numero para saber si es primo: \t");
    scanf("%d", &n);
```

```

while (di<n && pri !=1)
{
    if (n%di==0) pri=1;
    di++;
}
if (pri==0)
{
    printf("\n El numero %d es primo\t",n);
}else
{
    printf("\n El numero %d no es primo\n",n);
}
getchar();
main();
}
void mil(){
    for (int j = 2; j <=10000; j++){
        int a=0;
        for (int i = 1; i <=10000; i++)
        {
            if (j%i==0)
            {
                a++;
            }
        }
        if (a==2)
        {
            printf("\t %d",j);
        }
    }//fin del for
    getchar();
    main();
}

```

Prueba:

```

MINGW64:/c/Users/Leonel/Documents/UTM/Semestre1/ProgramacionEstructu...
Opciones
1) Si 'n' es primo
2) Saber los numero primos del 1 al 10mil
3)
4)salir
  ingresa una opcion:
1

  Ingrese un numero para saber si es primo:      7

  El numero 7 es primo  |

```

```

MINGW64:/c/Users/Leonel/Documents/UTM/Semestre1/ProgramacionEstructu...
Opciones
1) Si 'n' es primo
2) Saber los numero primos del 1 al 10mil
3)
4)salir
  ingresa una opcion: 2

      2      3      5      7      11      13      17      19      23
29      31      37      41      43      47      53      59      61      67
71      73      79      83      89      97      101      103      107      109
113     127     131     137     139     149     151     157     163     167
173     179     181     191     193     197     199     211     223     227
229     233     239     241     251     257     263     269     271     277
281     283     293     307     311     313     317     331     337     347
349     353     359     367     373     379     383     389     397     401
409     419     421     431     433     439     443     449     457     461
463     467     479     487     491     499     503     509     521     523
541     547     557     563     569     571     577     587     593     599
601     607     613     617     619     631     641     643     647     653
659     661     673     677     683     691     701     709     719     727
733     739     743     751     757     761     769     773     787     797
809     811     821     823     827     829     839     853     857     859
863     877     881     883     887     907     911     919     929     937
941     947     953     967     971     977     983     991     997     1009

```

## Ejercicio 5.35

terminar.

**5.35** Escriba un programa en C que juegue el juego de "adivine el número" como sigue: su programa escoge el número que se debe de adivinar seleccionando un entero al azar en el rango del 1 al 1000. El programa a continuación escribe:

```
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
```

El jugador entonces escribe su primera estimación. El programa responde con una de las siguientes:

### Algoritmo

1. Inicio
2. Ingresar un numero
3. Generar un numero aleatorio con rand.
4. Comparar los numeros
5. Si es el resultado, esta bien.

### Código

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int n, random;
int  nrandom();int x;

int main(){
    srand(time(NULL));
    x=rand()%1000 ;
    getchar();
    system("cls");
    printf("\n En que numero estoy pensado?[1-1000]");
    scanf("%d", &n);
    nrandom();
    return 0;
}

int  nrandom(){
    char op;
    printf("\n x=%d",x);
    if (n==x)
    {
```

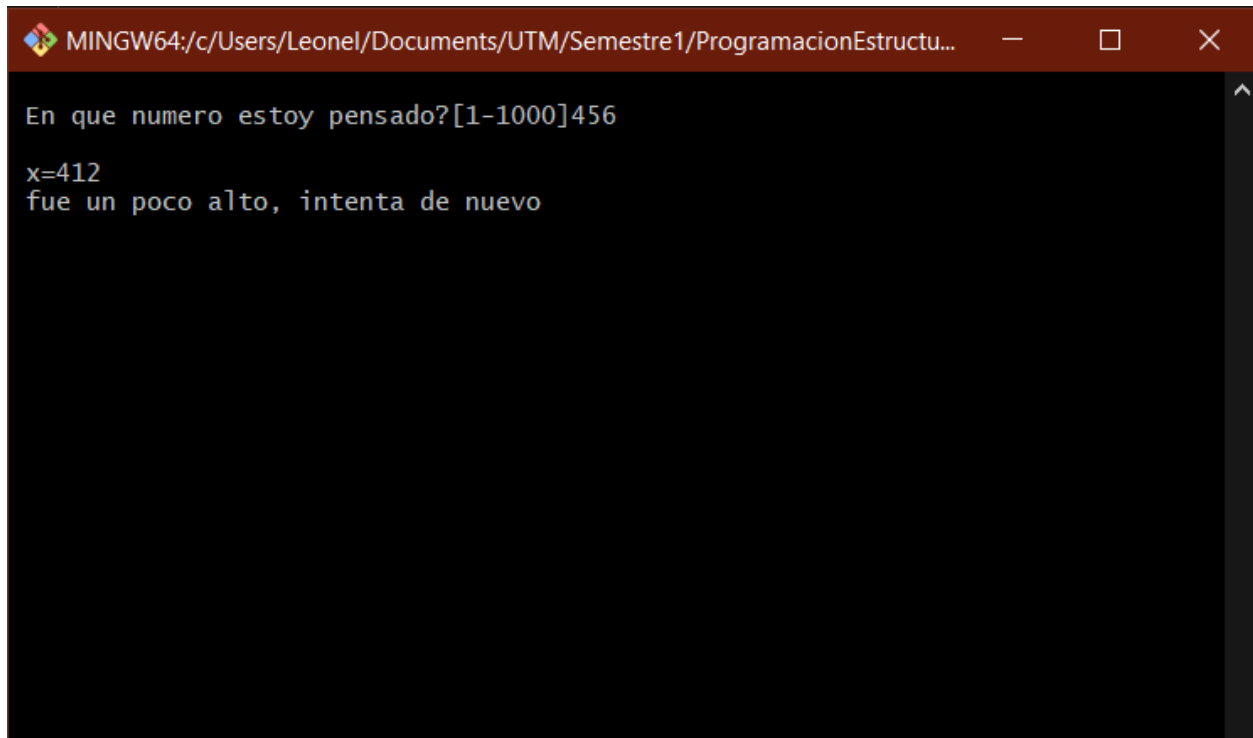


```

        printf("\n Excelente el numero era : %d\n quieres intentarlo de nuevo ?[y
o n]:", x );
        scanf("%s", &op);
        switch (op)
        {
            case 's': main();
            case 'n': return 0; break;
        }
    }else if (n>x)
    {
        printf("\n fue un poco alto, intenta de nuevo");
        getchar();
        main();
    }else if (n<x)
    {
        printf("\n fue un poco bajo, intenta de nuevo");
        getchar();
        main();
    }
    getchar();
}

```

Prueba:



```

MINGW64:/c/Users/Leonel/Documents/UTM/Semestre1/ProgramacionEstructu...
En que numero estoy pensando?[1-1000]456
x=412
fue un poco alto, intenta de nuevo

```

## Ejercicio 5.38

### 5.38 La serie Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21,...

empieza con los términos 0 y 1 y tiene la propiedad que cada término siguiente es la suma de los dos términos precedentes. a) Escriba una función *no recursiva* **fibonacci(n)** que calcule el número Fibonacci de orden **n**. b) Determine el número Fibonacci más grande que pueda ser impreso en su sistema. Modifique el programa de la parte a) para utilizar **double** en vez de **int**, a fin de calcular y regresar números Fibonacci. Deje que el programa cicle hasta que falle debido a valores en exceso altos.

Algoritmo:

1. Inicio
2. Pedimos la cantidad de números a mostrar
3. Es la suma de los 2 numeros anteriores hasta llegar al numero deseado
4. Se hace con un for
5. Se imprime cada ejecución
6. Fin

```

7. #include<stdlib.h>
8. #include<stdio.h>
9. int n, op;
10. void Fibo(), FiboDou(), Fibonum();
11. int main(){
12.     getchar();
13.     system("cls");
14.     printf("\n opciones\n 1) calcular la serie fibonacci de un nu
        mero\n 2)Fibonacci maximo\n 3) Fibonacci con double\n 4) salir \n
        opcion: ");
15.     scanf("%d", &op);
16.     do
17.     {
18.         switch (op){
19.             case 1: Fibo(); break;
20.             case 2: Fibonum();break;
21.             case 3: FiboDou();break;
22.             case 4: op =4; break;
23.             default: main(); break;
24.         }
25.     } while (op!=4);
26.
27.     return 0;
28. }
29.
30. void Fibo(){
31.     int a=0, b=1, c ;
32.     printf("\n Cuantos numero quieres crear ?");

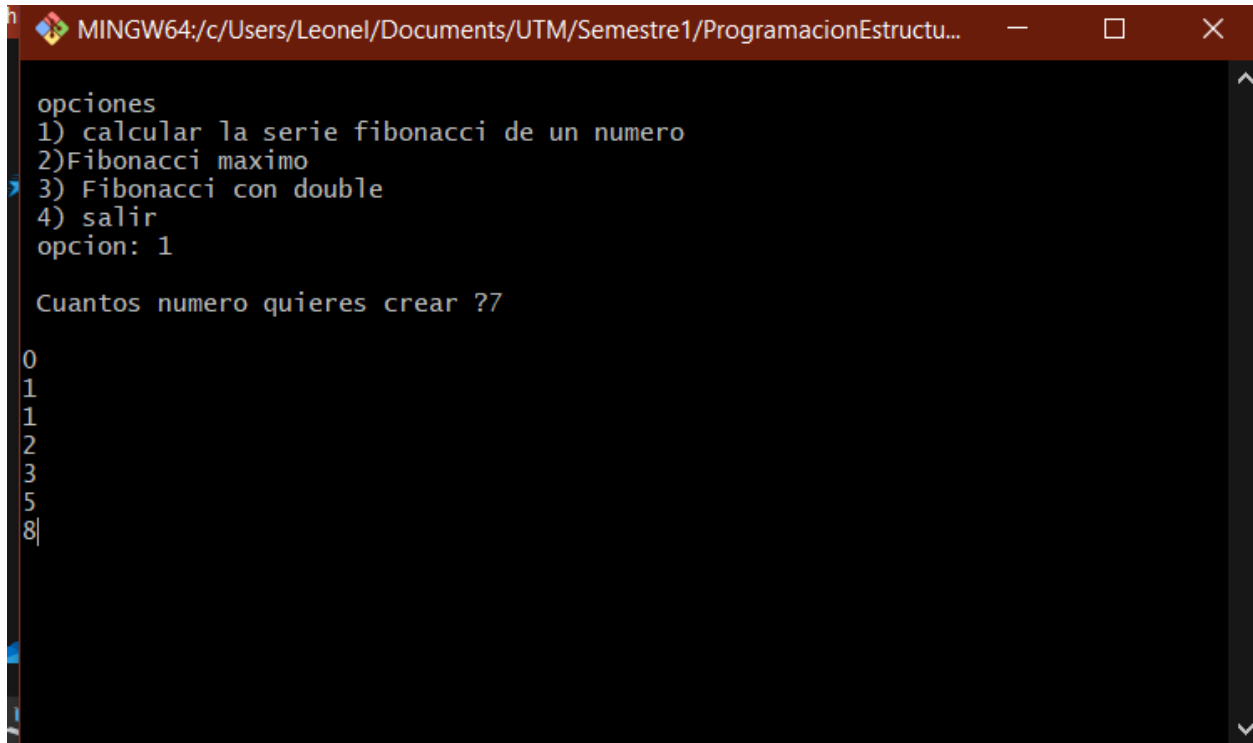
```

```

33.     scanf("%d", &n);
34.     printf("\n0\n1");
35.     for (int i = 0; i <=n-3; i++)
36.     {
37.         c = a + b;
38.         printf("\n%d",c);
39.         a=b;
40.         b=c;
41.     }
42.     getchar();
43.     main();
44.
45.}
46.void Fibonum(){
47.    int a=0, b=1, c, i ;
48.    printf("\n0\n1");
49.    for ( i = 0; i>=0 && c>0; i++)
50.    {
51.        c = a + b;
52.        printf("\n%d",c);
53.        a=b;
54.        b=c;
55.    }
56.    printf("\n De tipo entero solo se genera hasta %d", i);
57.    getchar();
58.    main();
59.}
60.void FiboDou(){
61.    double a=0, b=1, c, deli=0.0;
62.    int i, n ;
63.    printf("\n Ingresa la cantidad de numero que deseas generar:
    \t");
64.    scanf("%d", &n);
65.    printf("\n0.0\n1.0");
66.    for ( i = 0; i<=n ; i++)
67.    {
68.        c = a + b;
69.        printf("\n%.11f",c);
70.        a=b;
71.        b=c;
72.    }
73.    printf("\n De tipo entero solo se genera hasta %d", i);
74.    getchar();
75.    main();
76.}

```

Prueba:



```

MINGW64:/c/Users/Leonel/Documents/UTM/Semestre1/ProgramacionEstructu...
opciones
1) calcular la serie fibonacci de un numero
2) Fibonacci maximo
3) Fibonacci con double
4) salir
opcion: 1

Cuantos numero quieres crear ??
0
1
1
2
3
5
8|
  
```

### Ejercicios 5.39

**5.39** (*Torres de Hanoi*) Todos los científicos de cómputo incipientes deben de enfrentarse con ciertos problemas clásicos, y las Torres de Hanoi (vea la figura 5.18) es uno de los más famosos. Dice la leyenda que en un templo del Lejano Este, los monjes están intentando mover una pila de discos de una estaca hacia otra. La pila inicial tenía 64 discos ensartados en una estaca y acomodados de la parte inferior a la superior en tamaño decreciente. Los monjes están intentando mover la pila de esta estaca a la segunda con las limitaciones que exactamente un disco debe de ser movido a la vez, y en ningún momento se puede colocar un disco mayor por encima de un disco menor. Existe una tercera estaca disponible para almacenamiento temporal de discos. Se supone que cuando los monjes terminen su tarea llegará el fin del mundo, por lo cual para nosotros existe poca motivación en ayudarles en sus esfuerzos.

Supongamos que los monjes están intentando mover los discos de la estaca 1 a la estaca 3. Deseamos desarrollar un algoritmo que imprima la secuencia precisa de las transferencias disco a disco entre estacas.

Si fuéramos a enfocar este problema con métodos convencionales, nos encontraríamos rápidamente enmarañados y sin esperanza de poder manejar los discos. En vez de ello, si atacamos el problema teniendo en mente la recursión, de inmediato se vuelve manejable. El mover  $n$  discos puede ser visualizado en términos de sólo mover  $n-1$  discos (y de ahí la recursión), como sigue:

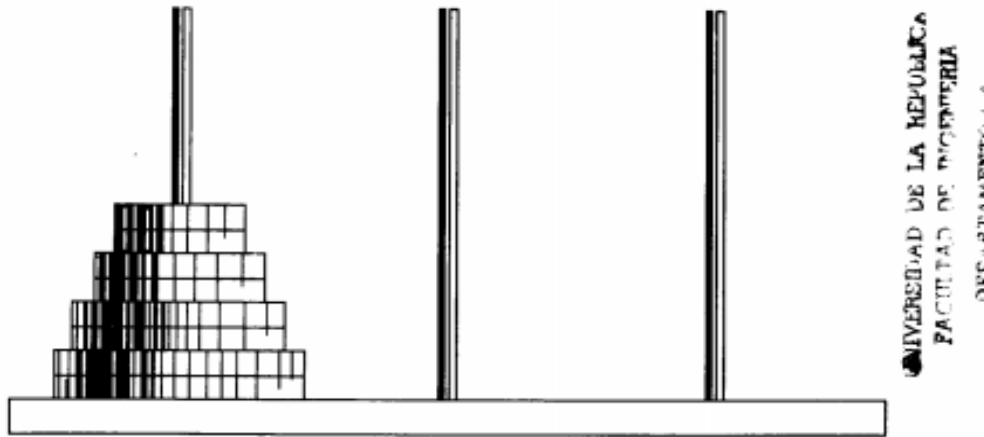


Fig. 5.18 Las Torres de Hanoi para el caso de cuatro discos.

1. Mover  $n-1$  discos de la estaca 1 a la estaca 2, utilizando a la estaca 3 como un área de almacenamiento temporal.
2. Mover el último disco (el más grande) de la estaca 1 a la estaca 3.
3. Mover los  $n-1$  discos de la estaca 2 a la estaca 3, utilizando la estaca 1 como área de almacenamiento temporal.

El proceso termina cuando la última tarea consiste en mover el disco  $n = 1$ , es decir, el caso base. Esto se lleva a cabo en forma trivial moviendo el disco, sin necesidad de utilizar el área temporal de almacenamiento.

Escriba un programa para resolver el problema de las Torres de Hanoi. Utilice una función recursiva con cuatro parámetros:

1. El número de discos a moverse
2. La estaca en la cual se acumularán estos discos al inicio
3. La estaca a la cual esta pila de discos se moverá
4. La estaca a utilizarse como área de almacenamiento temporal

Su programa deberá imprimir las instrucciones precisas que deberán seguirse para mover los discos de la estaca de arranque a la estaca destino. Por ejemplo, para mover una pila de tres discos de la estaca 1 a la estaca 3, su programa deberá imprimir la serie siguiente de movimientos:

```
1 → 3 (Esto significa mover un disco de la estaca 1 a la estaca 3)
1 → 2
3 → 2
1 → 3
2 → 1
2 → 3
1 → 3
```

### Código

```
#include <stdio.h>
#include <stdlib.h>

void TorresHanoi();
int main(){
    int n;
    printf("\nCon Cuantos discos cuenta? :");
    scanf("%d", &n);
    TorresHanoi(n ,1,3,2);

    getchar();
    return 0;
}
```

```

}

void TorresHanoi(int n, int o, int d, int aux){
    if (n>0)
    {
        TorresHanoi(n-1, o, aux, d);
        printf("\n  %d ---> %d",o,d);
        TorresHanoi(n-1, aux, d, o);
    }
}

```

### Prueba:

```

Leonel@MSI MINGW64 ~/Documents/UTM/Semestre1/ProgramaciónEstructurada/codigos/EjerciciosDeDietel/Ejercicios5_0 (master)
$ ./5_39.exe

Con Cuantos discos cuenta? :3

1 ---> 3
1 ---> 2
3 ---> 2
1 ---> 3
2 ---> 1
2 ---> 3
1 ---> 3

```

### Conclusiones:

Las funciones como programador las tendremos que llevar durante toda nuestra vida laboral, ya que es de las funciones principales, a un tengo algunas dudas con la recursividad pero necesito investigar a un mas, en mi experiencia ahorita mismo me hace falta bastante conocimiento, ya avance bastante a través de las clases, lo aprendido nos muestra a como Se puede llamar una función para mandar a traer algún proceso que no queremos que ocupe espacio en el main. No eh presentado problemas de aprendizaje ya que aprendí a programar a través de cursos de internet ( Platz.com) y me ayudo un poco a indagar y a investigar por mi cuenta. Gracias por sus clases.