Instituto Politécnico Nacional
La Técnica al Servicio de la Patria

**Unidad Profesional Interdisciplinaria de Ingeniería Campus Tlaxcala UPIIT**

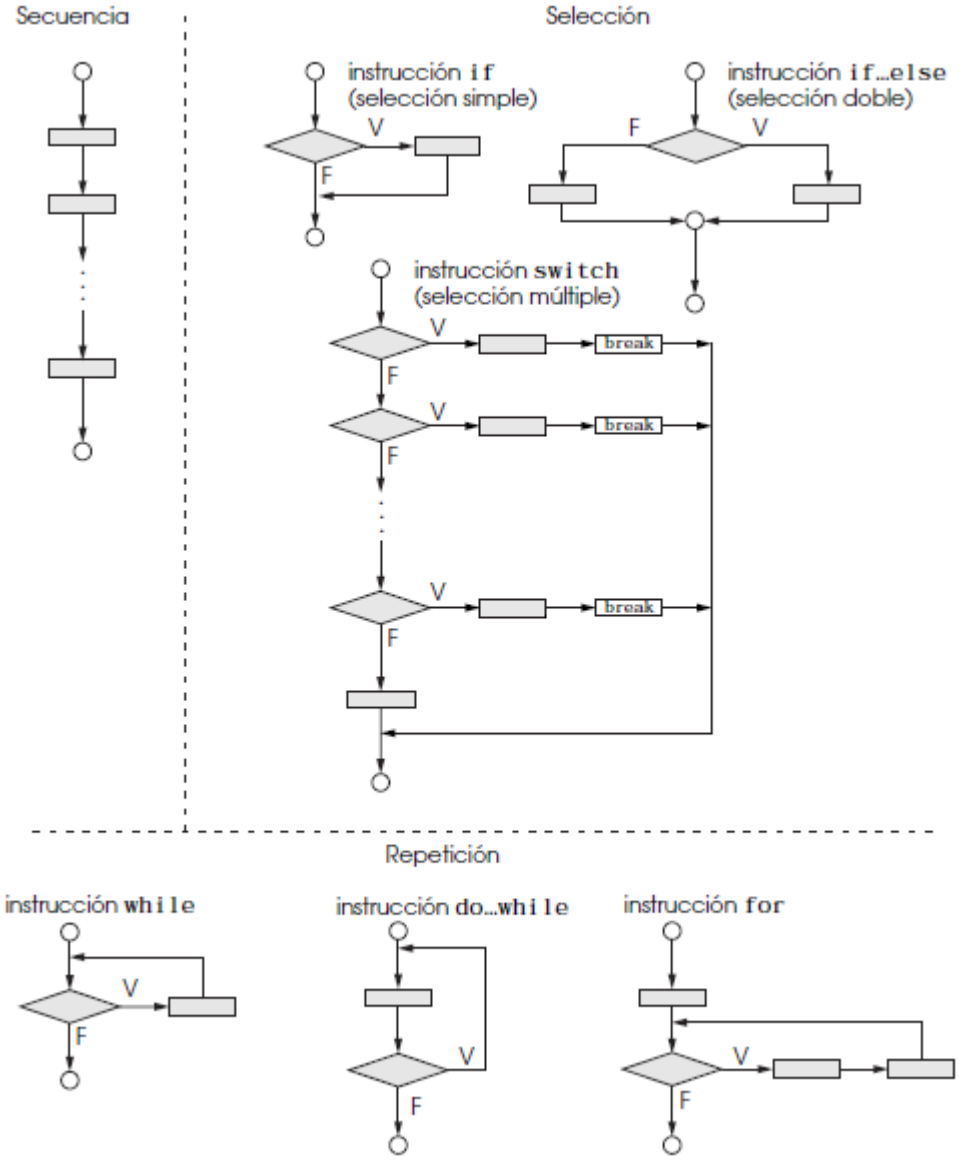
# Fundamentos de Programación

Esaú Eliezer Escobar Juárez

# Programación estructurada

- Esta ligada al control de ejecución
  - Las instrucciones se ejecutan una tras de otra.
  - Unas partes se ejecutan, otras no dependiendo de las condiciones.
  - También tenemos ciclos que se repiten de acuerdo a un número fijo o a una condición determinada.

# Estructuras de control

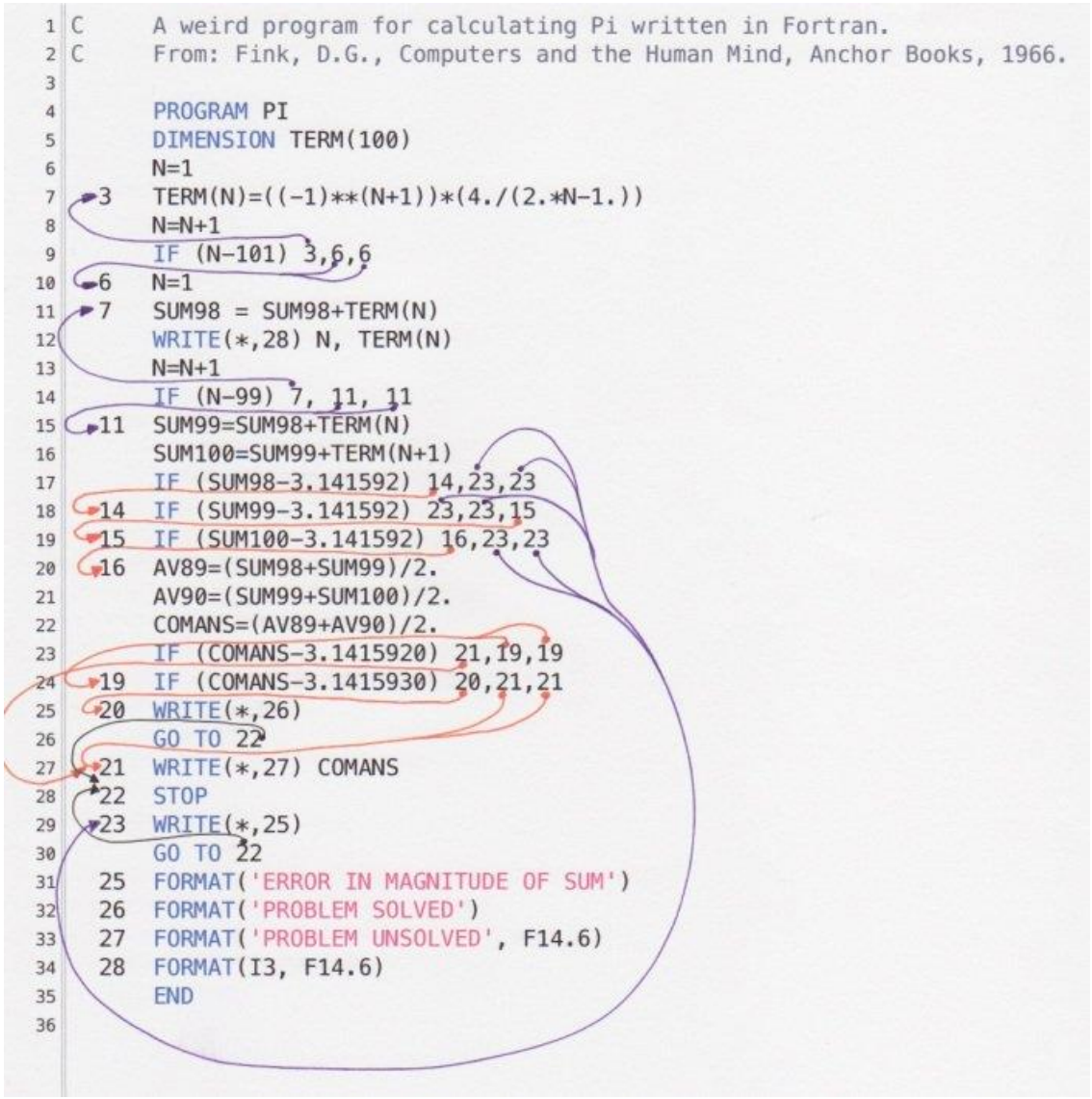


# Instrucción goto

- Programación antigua (Fortran).
- Hacen a los programas poco legibles.

*Una de las maldiciones que imponían algunos de los primeros lenguajes de programación era la que se bautizó como código espagueti ('spaghetti code'). La razón, su falta de reglas de programación, la complejidad de su control de flujo y esa analogía con ese montón de hilos intrincados y anudados que forman los espagueti.*

# Instrucción goto



```
C     A weird program for calculating Pi written in Fortran.
C     From: Fink, D.G., Computers and the Human Mind, Anchor Books, 1966.

      PROGRAM PI
      DIMENSION TERM(100)
      N=1
    3 TERM(N)=((-1)**(N+1))*(4./(2.*N-1.))
      N=N+1
      IF (N-101) 3,6,6
    6 N=1
    7 SUM98 = SUM98+TERM(N)
      WRITE(*,28) N, TERM(N)
      N=N+1
      IF (N-99) 7, 11, 11
   11 SUM99=SUM98+TERM(N)
      SUM100=SUM99+TERM(N+1)
      IF (SUM98-3.141592) 14,23,23
   14 IF (SUM99-3.141592) 23,23,15
   15 IF (SUM100-3.141592) 16,23,23
   16 AV89=(SUM98+SUM99)/2.
      AV90=(SUM99+SUM100)/2.
      COMANS=(AV89+AV90)/2.
      IF (COMANS-3.1415920) 21,19,19
   19 IF (COMANS-3.1415930) 20,21,21
   20 WRITE(*,26)
      GO TO 22
   21 WRITE(*,27) COMANS
   22 STOP
   23 WRITE(*,25)
      GO TO 22
   25 FORMAT('ERROR IN MAGNITUDE OF SUM')
   26 FORMAT('PROBLEM SOLVED')
   27 FORMAT('PROBLEM UNSOLVED', F14.6)
   28 FORMAT(I3, F14.6)
      END
```

# Programación **sin** goto

- A finales de los 60 surge la programación estructurada.

**Teorema (Edsger Dijkstra)**

Todo programa puede escribirse utilizando únicamente tres estructuras de control básicas:

- **Secuencia, selección e iteración**

- Puede ser leído en secuencia

- Predominante hasta la POO

## Reglas para crear programas estructurados

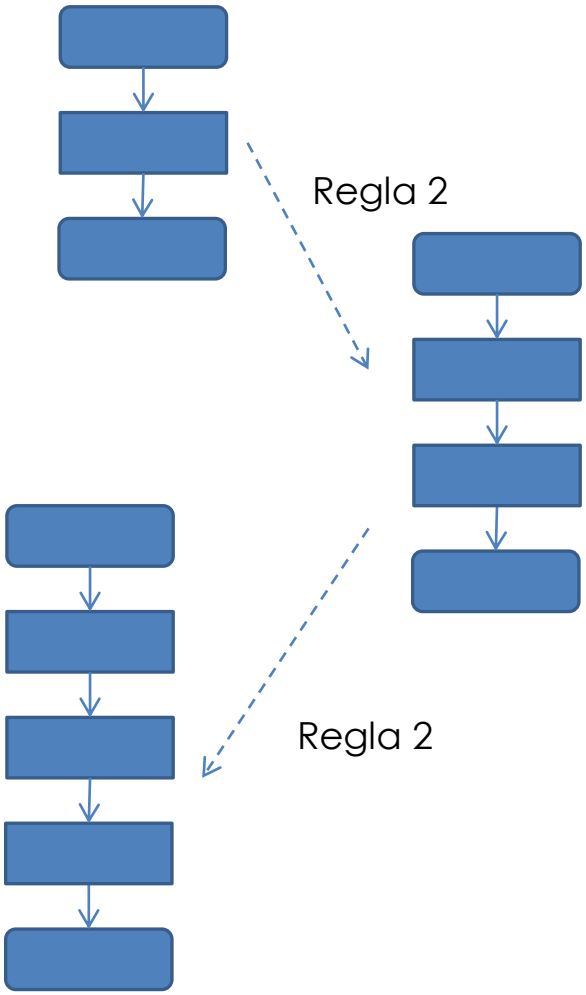
1) Comience con el "diagrama de flujo más sencillo".

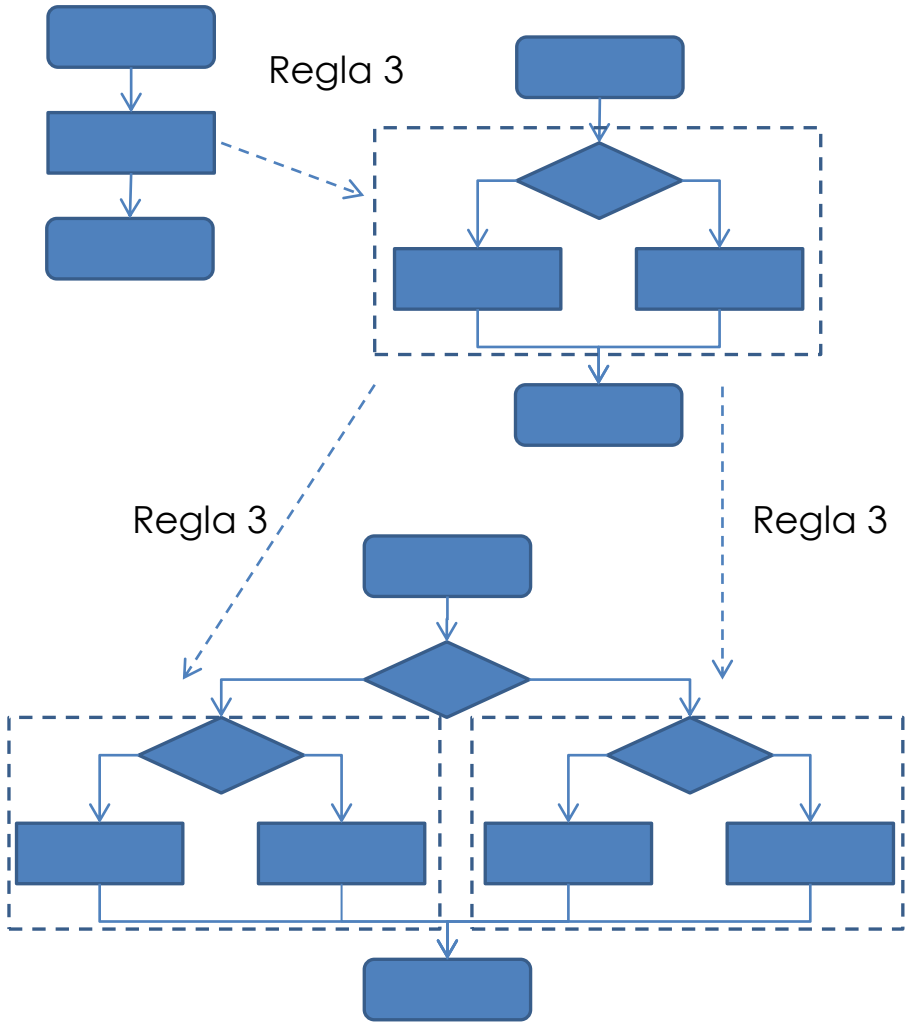
2) Cualquier rectángulo (acción) puede ser reemplazada por dos rectángulos (acciones) en secuencia.

3) Cualquier rectángulo (acción) puede ser reemplazado por cualquier instrucción de control (secuencia, **if**, **if…else**, **switch**, **while**, **do…while** o **for**).

4) Las reglas 2 y 3 pueden aplicarse con tanta frecuencia como desee, y en cualquier orden.

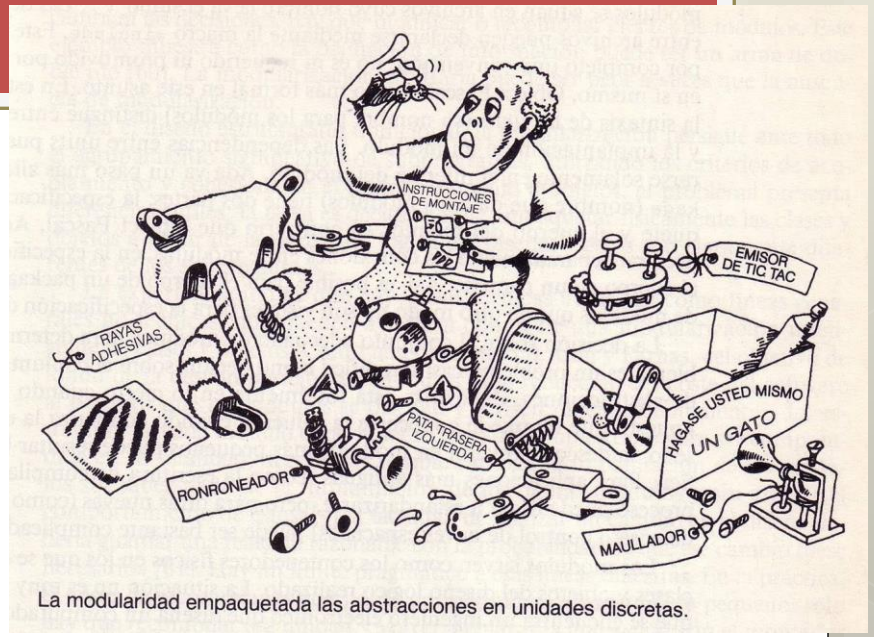
# Reglas para crear programas estructurados



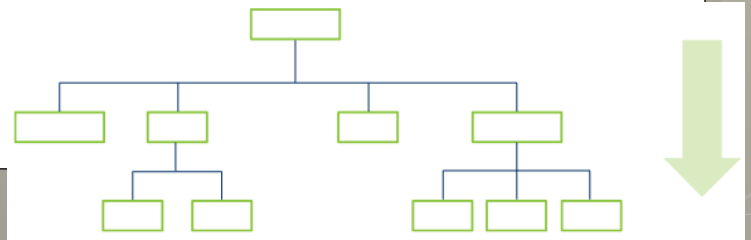
# Programas estructurados modernos

- Segmentos o módulos
  - Una entrada, una salida.
  - Sin bucles infinitos.
  - Sin secciones que no se lleguen a ejecutar.



La modularidad empaquetada las abstracciones en unidades discretas.

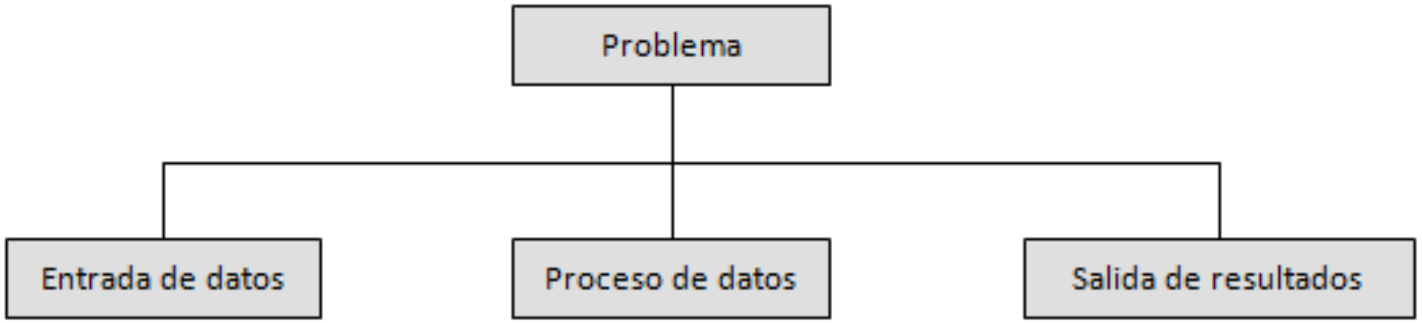
- A los módulos se les puede combinar con las 3 técnicas básicas de control: **Secuencia, selección e iteración.**
- Una correcta partición del problema producirá una nula o casi nula dependencia entre los módulos
- Cada módulo se puede trabajar de forma independiente
- Enfoque top-down
  - División sucesiva del problema en subproblemas



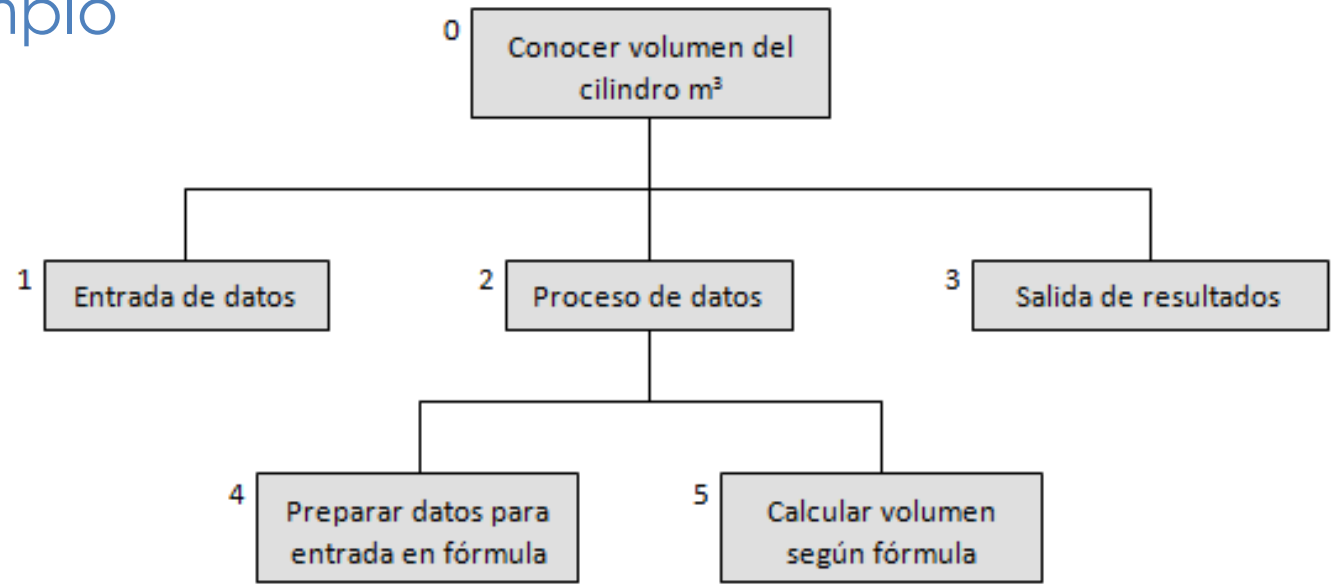
# Ventajas de la modularidad

- Reducir la complejidad del problema
- Reducir el tamaño del problema
- Favorecer el entendimiento del problema
- Facilitar la cooperación entre programadores
- Reutilizar código.
- Facilitan la lectura del código.
- Ayuda a ser más clara la lógica del programa.
- Protege contra efectos colaterales (destrucción accidental de datos de programa).
- Permite plantear una solución completa del problema, para luego profundizar en los detalles.
- La depuración es más fácil de realizar ya que primero se corrigen errores en los módulos de nivel inferior..

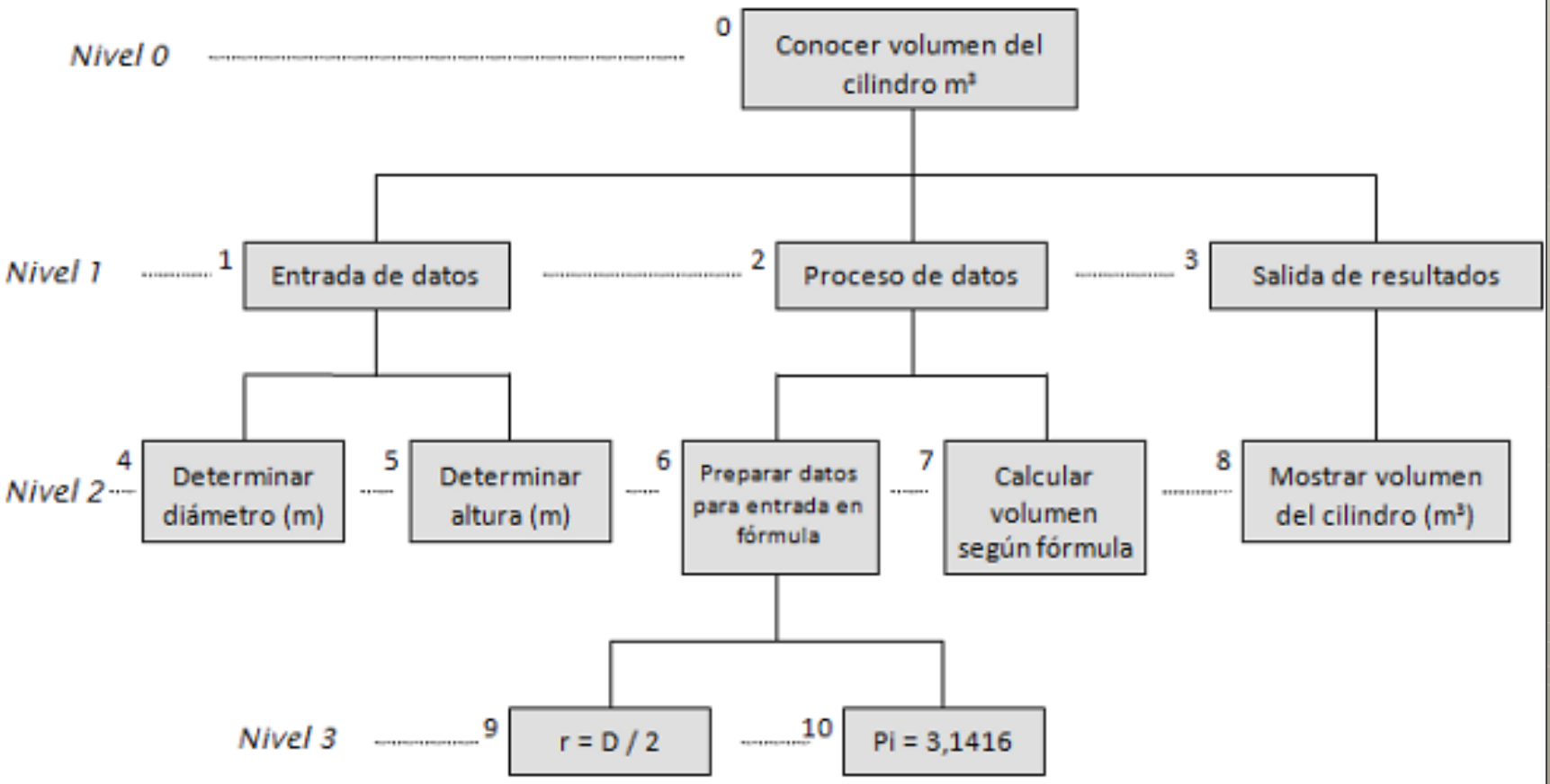
# Enfoque descendente top-down




# Ejemplo



# Ejemplo


Nivel 0
0
Conocer volumen del cilindro m³
Nivel 1
1
Entrada de datos
2
Proceso de datos
3
Salida de resultados
Nivel 2
4
Determinar diámetro (m)
5
Determinar altura (m)
6
Preparar datos para entrada en fórmula
7
Calcular volumen según fórmula
8
Mostrar volumen del cilindro (m³)
Nivel 3
9
r = D / 2
10
Pi = 3,1416

# Arreglos

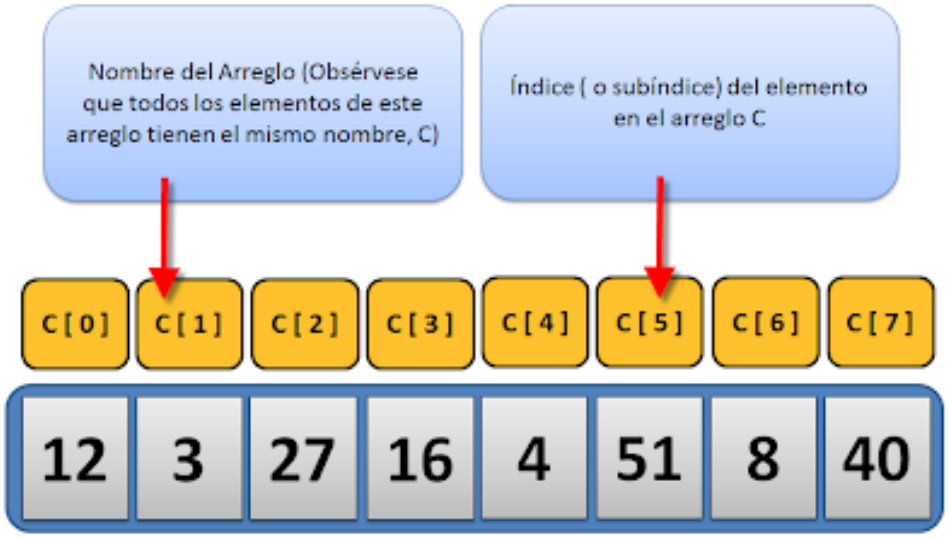
- Grupo consecutivo de localidades de memoria relacionadas por el nombre y el tipo.
- Colección o grupo de datos
- Cada dato tiene su posición.
- Todos los datos son del mismo tipo

  **No existen arreglos mezclados**

- Podemos visualizarlos como un grupo de cajas, una detrás de otra.

# Arreglos

- Cada arreglo tiene
  - Tamaño: Cuantos datos va a contener
  - Tipo: **todos** los datos tienen el mismo
  - Nombre: un solo nombre para todos los datos




- Para acceder a una localidad del arreglo se utilizan el número de posición entre corchetes [ ].

  c [ 4 ]  c [ 8 ]

# Declarar Arreglos

- Se usa

```
tipo nombre[n_elementos]
```

- Por ejemplo

```
int c[12]
```

- Para inicializar un arreglo tenemos que hacerlo elemento a elemento o con una lista de inicialización.

# Declarar Arreglos

- Uso de constantes simbólicas.

  #define TAMANIO 7

- Se sustituyen por su valor al compilar.

- No utilizan un espacio de memoria.

Dev C++ 14

# Función rand( )

- Permite generar números pseudoaleatorios
- Biblioteca stdlib.h
- Entre 0 y RAND_MAX con distribución uniforme.
- RAND_MAX = 32767 en 16 bits

# Función rand( )

- El número requerido en los programas a menudo difiere del proporcionado por rand
- Se realiza un escalamiento utilizando %
- Por ejemplo: Generar números aleatorios para simular el tiro de un dado

```
(rand()%6)+1;
```

Dev C++ 15

# Función rand( )

- Porque se llama pseudoaleatorios.
- ¿Cómo podemos evitarlo?
- Función srand(semilla) permite establecer una semilla con el tipo unsigned int.
- Semilla automática:

```
srand( time( NULL ) );
```

Dev C++ 16

# Cadena de caracteres

- Los arreglos de tipo char

```
char cadena[] = "primero";
```

***Equivalentes***

```
char cadena[] = {'p','r','i','m','e','r','o','\0'};
```

- Podemos leer una cadena con scanf usando el especificador de formato %s.

```
char cadena[20];
scanf("%s",cadena);
```

Notar que no hay &

# Cadena de caracteres

- Podemos mostrar una cadena con printf usando el especificador de formato %s.

```
printf("%s",cadena);
```

- Tanto scanf como printf no verifican el tamaño del arreglo, leen o imprimen hasta encontrar el carácter '\0'.

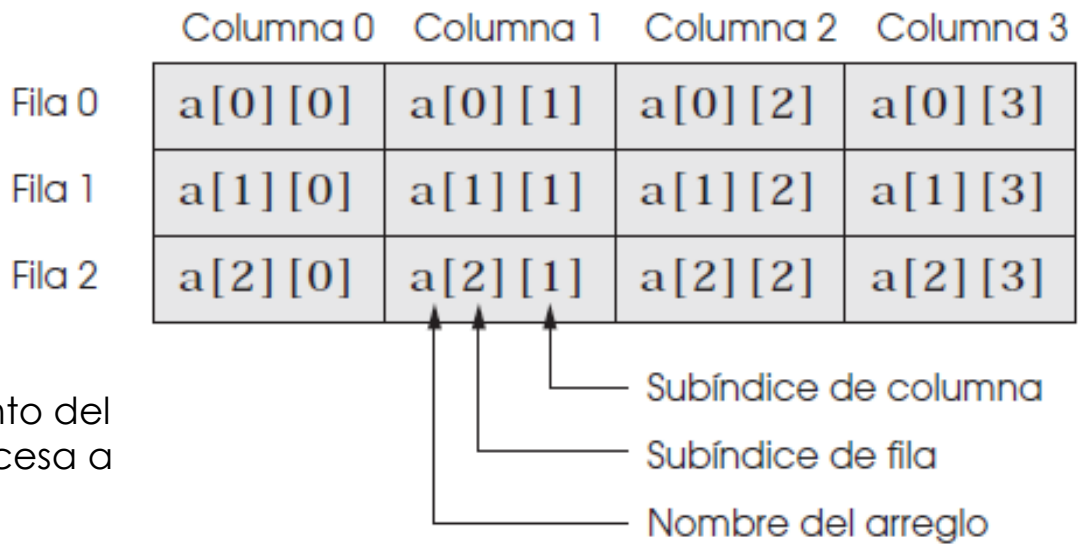
- Otra función útil para leer cadenas de caracteres es gets.

# Arreglos con múltiples subindices

- Un arreglo se puede declarar con varios subindices:

```
int a[3][4]; //Arreglo de 3x4
```

- Un arreglo con m filas y n columnas se denomina arreglo de m x n.




Cada elemento del arreglo se accesa a través de los subindices.

# Arreglos con múltiples subindices

- Un arreglo se puede inicializar en su declaración, ejemplo:

```
int b[2][2] = {{1,2},{3,4}}; //Arreglo de 2x2
```

- Los arreglos pueden tener mas de 2 dimensiones.