John Wilder Tukey

# Resolución de Problemas y Algoritmos

**Clase 5**
**Repetición incondicional**
**(sentencia FOR)**

**Dr. Alejandro J. García**
http://cs.uns.edu.ar/~ajg

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca - Argentina

## Conceptos de las 4 clases anteriores

- **Algoritmo. Primitiva. Traza.**
- **Computadora. Hardware. Bit. Byte. Memoria RAM.**
- **Lenguaje de programación. Programa. Código fuente.**
- **Sintaxis de un lenguaje. Diagrama sintáctico.**
- **Pascal:**
  - **Identificadores reservados y predefinidos**
  - **Constantes, variables y tipos de datos.**
  - **Primitivas: asignación (:=) read, readln, write, writeln**
  - **Tipos predefinidos: real, integer, char, boolean.**
  - **Expresiones. Operaciones y funciones predefinidas.**
  - **Expresiones lógicas. Expresiones equivalentes.**
  - **Sentencia condicional IF-THEN-ELSE**

**¿Preguntas?**



## Programación de computadoras a bajo nivel

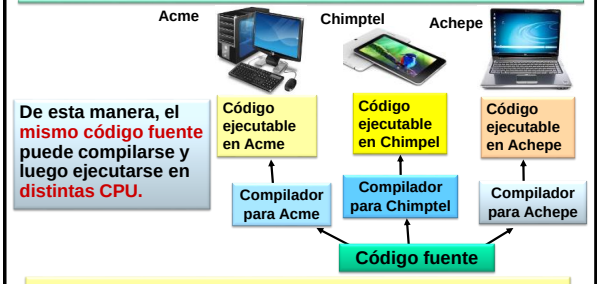
Acme Chimtel Maple Achepe

- **Cada CPU de una computadora es capaz de ejecutar un único lenguaje llamado lenguaje máquina.**
- **Generalmente, cada marca de CPU tiene su propio lenguaje máquina.**

**¿Tengo que aprender el lenguaje máquina de cada CPU?**
**Afortunadamente NO (como verá a continuación)**



## Programación de computadoras a alto nivel




Un **compilador** permite traducir el código fuente de un programa, a otro lenguaje (típicamente lenguaje de máquina) permitiendo generar **código ejecutable**.



## Conceptos: compilación

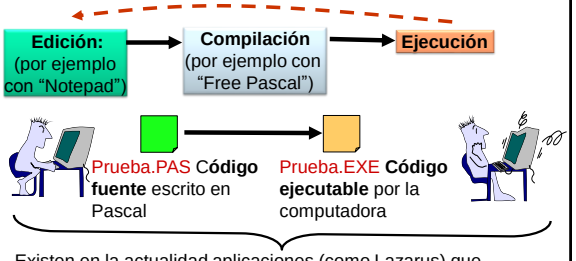
Un **compilador** es un programa que traduce un programa, que está escrito en un lenguaje de programación, a otro lenguaje de programación, generando un programa equivalente.

- Este proceso de traducción se conoce como **compilación**.
- Un compilador permite traducir el código fuente de un programa, a otro lenguaje (típicamente lenguaje de máquina) permitiendo generar código ejecutable.

El **código ejecutable** es una secuencia de código que la computadora puede ejecutar directamente al ser invocado, (sin necesidad que el compilador esté presente.) Generalmente de extensión EXE o COM.



## Edición / Compilación / Ejecución




Existen en la actualidad aplicaciones (como Lazarus) que brindan un **entorno de programación**, donde se puede editar, compilar (con Free Pascal) y ejecutar programas en Pascal dentro del mismo entorno.

## Elementos predefinidos

- **La definición de un lenguaje de programación es algo teórico. A la definición original se la llama Estándar.**
- **Los elementos predefinidos (ej el tipo INTEGER) son generalmente propuestos en la definición del lenguaje y luego provistos por el compilador (ej. Free Pascal).**
- **Así en un lenguaje de programación puede haber tipos predefinidos, constantes predefinidas, primitivas predefinidas, operaciones o funciones predefinidas.**
- **Las organizaciones o compañías que implementan un compilador deben respetar a la definición estándar.**
- **Pero muchas veces estos compiladores agregan elementos predefinidos y extienden al estándar. Como por ejemplo el tipo LONGINT en Free Pascal.**



## Software [Wikipedia]

«Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación»

[Extraído del estándar 729 del IEEE]

Considerando esta definición, el concepto de ***software*** va más allá de los programas de computación; también su documentación, los datos a procesar e incluso la información de usuario forman parte del software: es decir, *abarca todo lo intangible*, todo lo «no físico» relacionado; en contraposición a los componentes físicos, que son llamados ***hardware***.



## Conceptos: Software [Wikipedia]

***Software*** es una palabra inglesa (literalmente: partes blandas o suaves); como en español no posee una traducción adecuada, fue admitida por la Real Academia Española que lo define como: *«Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora ».*

Se conoce como ***software*** *al equipamiento lógico o soporte lógico* de un sistema informático; comprende el conjunto de los componentes **lógicos** necesarios que hacen posible la realización de tareas específicas.



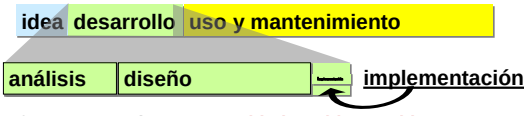
## Historia [Wikipedia]

- John W. Tukey usó el termino "Software de Computación" (Computer Software) en un artículo de 1958 en el *American Mathematical Monthly*, aparentemente fue el primer uso de este término.
- Además, mientras trabajaba con John von Neumann en los primeros diseños de las primeras computadoras, Tukey introdujo la palabra "bit" como acrónimo de las palabras *Binary Digit* (“Dígito binario”).

**John Wilder Tukey**



## Software: ciclo de vida




- **El software surge de una necesidad, problema o idea.**
  Por ejemplo: páginas web para la materias del DCIC.
- **Luego se desarrolla.**
- **Finalmente se usa durante un tiempo y de ser necesario se modifica. (según la experiencia el 60% del ciclo de vida lo constituye esta etapa)**
- **Es importante utilizar metodologías y técnicas que simplifiquen el desarrollo y faciliten el mantenimiento (algunas de ellas vamos a comenzar a ver en RPA)**



## Etapas en el desarrollo de software

**desarrollo**

DOCUMENTACIÓN ↓ ↑

1. **Definición de requerimientos**
2. **Análisis del sistema**
3. **Diseño de sistemas.**
4. **Implementación de programas.**
5. **Pruebas unitarias (programas).**
6. **Pruebas de integración.**
7. **Entrega del sistema y Mantenimiento.**

En todas las etapas debe realizarse una documentación

## RPA en las etapas del desarrollo de software

RPA

1. Definición de requerimientos
2. Análisis del sistema
3. Diseño de sistemas.
4. Implementación de programas.
5. Pruebas unitarias (programas).
6. Pruebas de integración.
7. Entrega del sistema y Mantenimiento.



## El desarrollo de software y algunas materias

RPA

1. Definición de requerimientos — ADS
2. Análisis del sistema
3. Diseño de sistemas. — IPOO – TdP – DDS
4. Implementación de programas. — IPOO-ED-TdP
5. Pruebas unitarias (programas).
6. Pruebas de integración. — APS
7. Entrega del sistema y Mantenimiento.



## Conceptos: Pautas para un buen estilo de programación

- En la vida profesional del desarrollo de software, es sumamente importante que un programa (o algoritmo) pueda ser **entendido rápidamente** por una **persona** que lo tiene que leer.
- Para lograr esto, al escribir programas (por más modestos que sean), hay que seguir ciertas **pautas** de "**buen estilo de programación**".
- En las evaluaciones de RPA tendremos en cuenta las siguientes pautas de buena programación:

1. **Usar tipos de datos adecuados.**
2. **Uso de nombres representativos en identificadores.**
3. **Indentación (del inglés "indent").**
4. **Comentarios en el código fuente.**

{…entre llaves…} // o al finalizar una línea



## Buen estilo de programación: tipos de datos apropiados

Tener tipos de datos para las variables permite claridad y abstracción: dos conceptos fundamentales en el desarrollo, mantenimiento y futuras actualizaciones del software. Usar nombres representativos ayuda a comprender el uso que se le da a un dato en un programa.

Ejemplos :

| Dato a representar | Nombre de la variable | Tipo de dato predefinido |
|---|---|---|
| Sueldo de un empleado | Sueldo_empleado | REAL |
| Letra inicial del apellido | Inicial_apellido | CHAR |
| Si es o no es año bisiesto | Es_bisiesto | BOOLEAN |
| Año de nacimiento | Anio_nacimiento | INTEGER |



## Uso de nombres representativos

☹

```
IF (x1 > x2)
THEN
IF (x1 > w)
THEN z8:= x1
ELSE  z8 := w
ELSE IF (x2 > w)
THEN z8 := x2
ELSE  z8 := w
```

😐

```
IF (num1 > num2)
THEN
IF (num1 > num3)
THEN maximo := num1
ELSE maximo := num3
ELSE IF (num2 > num3)
THEN maximo := num2
ELSE maximo := num3
```



## Indentación

😐

```
IF (num1 > num2)
THEN
IF (num1 > num3)
THEN maximo := num1
ELSE maximo := num3
ELSE IF (num2 > num3)
THEN maximo := num2
ELSE maximo := num3
```

☺

```
IF (num1 > num2)
THEN
   IF (num1 > num3)
      THEN maximo := num1
      ELSE maximo := num3
ELSE
   IF (num2 > num3)
      THEN maximo := num2
      ELSE maximo := num3
```

## Comentarios en el código

😐

```
IF (num1 > num2)
THEN
   IF (num1 > num3)
      THEN maximo := num1
      ELSE maximo := num3
ELSE
   IF (num2 > num3)
      THEN maximo := num2
      ELSE maximo := num3
```

☺

```
{ calcula el máximo entre 3
  números: num1, num2 y num3}
IF (num1 > num2)
THEN
   IF (num1 > num3)
      THEN maximo := num1
      ELSE maximo := num3
ELSE //  num1 <= num2
   IF (num2 > num3)
      THEN maximo := num2
      ELSE maximo := num3
{… la variable máximo ahora tiene
   el mayor valor de los 3….}
```



## Compare…

☹

```
IF (x1 > x2)
THEN
IF (x1 > w)
THEN z8:= x1
ELSE  z8 := w
ELSE IF (x2 > w)
THEN z8 := x2
ELSE  z8 := w
```

☺

```
{ calcula el máximo entre 3
  números: num1, num2 y num3}
IF (num1 > num2)
THEN
   IF (num1 > num3)
      THEN maximo := num1
      ELSE maximo := num3
ELSE // num1 <= num2
   IF (num2 > num3)
      THEN maximo := num2
      ELSE maximo := num3
{… la variable máximo ahora tiene
   el mayor valor de los 3….}
```



## En papel…

```
{ calcula el máximo entre 3   números: num1, num2 y num3}
IF (num1 > num2)
THEN
    IF (num1 > num3)
       THEN máximo := num1
       ELSE máximo := num3
ELSE // num1 <= num2
    IF (num2 > num3)
       THEN máximo := num2
       ELSE máximo := num3
{… la variable máximo ahora tiene  el mayor valor de los 3….}
```

☺

Si está trabajando sobre un papel o pizarrón (práctico, examen, etc ), también puede usar líneas demarcadoras



## Repeticiones en algoritmos

**Para especificar un algoritmo es posible usar:**

**(1) Secuencia**

**(2) Condiciones**

**(3) Repeticiones:** permiten especificar de una manera abreviada una secuencia repetida de operaciones.

```
Llenar botella
Pasar a bidón
Llenar botella
Pasar a bidón
Llenar botella
Pasar a bidón
Guardar bidón
```

⟺

```
Repetir 3 veces:
    Llenar botella
    Pasar a bidón
Guardar bidón
```

**Veremos a continuación como programar en Pascal para que una sentencia se repita un número fijo de veces.**



## Conceptos: tipos ordinales en Pascal

**De los 4 tipos simples predefinidos que hemos visto, INTEGER,CHAR y BOOLEAN son tipos ordinales (REAL no es ordinal).**

**Los tipos ordinales tienen estas características:**

- **Tienen un primer y último elemento.**
- **Tienen un orden, y para cada elemento está definido el siguiente** (a excepción del último) **y el anterior** (a excepción del primero).

**Esto permite utilizar sobre los tipos ordinales las operaciones predefinidas:**

- **succ() retorna el siguiente (excepto del último)**
- **pred() retorna el anterior (excepto del primero)**
- **ord() retorna el número de orden**



## Concepto: repetición incondicional en Pascal

**Las sentencias de un ciclo FOR-TO se ejecutan CERO o más veces dependiendo de *valor_inicial* y *valor_final*.**

**La sentencia se ejecuta un número fijo de veces y luego continua en: ……**

```
FOR V:= valor_inicial TO valor_final
DO
    1 sentencia simple
    o compuesta ;
Otra sentencia siguiente;
```

- **V debe ser una variable de tipo ordinal (que se suele llamar variable de control)**
- ***valor_inicial* y *valor_final* son expresiones cuyo valor resultante debe pertenecer al mismo tipo que la variable V.**
- **Al comenzar a V se le asigna *valor_inicial*.**
- **Luego, V es incrementada automáticamente de a uno en cada repetición (hasta llegar a *valor_final*).**

## Ejemplo (p)

Escriba un programa para mostrar por pantalla todos los números enteros entre 1 y 5.

```
PROGRAM ListaNumeros;
{Muestra todos los números enteros entre 1 y 5}
VAR num:INTEGER;
BEGIN
writeln('Números');
FOR num:= 1 TO 5
    DO writeln(num);
writeln('enter para continuar');
readln; // mantiene abierta consola
END.
```

```
Números
1
2
3
4
5
enter para continuar
```

| num |
|---|
| ? |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| |



## Repetición incondicional

```
FOR V:= valor_inicial TO valor_final
DO sentencia simple o compuesta
```

- *valor_inicial* y *valor_final* son expresiones cuyo valor debe pertenecer al mismo tipo que la variable de control V.
- La *sentencia* (que puede ser compuesta), se repetirá un número fijo de veces: *valor_final* – *valor_inicial* + 1.

```
FOR V:= 1 TO 5 DO writeln(V);
```
← repite 5 veces

- Si *valor_final* es menor estricto a *valor_inicial* entonces se repetirá 0 veces.

```
FOR V:= 5 TO 1 DO writeln(V);
```
← repite 0 veces



## Repetición incondicional

- Al comenzar a V se le asigna el valor inicial y luego, V se incrementada automáticamente de a uno hasta llegar al valor final.

```
FOR V:= 1 TO 100
DO <sentencia>
```
Aquí <sentencia> se repite 100 veces

```
FOR V:= 100 TO 199
DO <sentencia>
```
Aquí <sentencia> se repite 100 veces

```
FOR V:= -10 TO -1
DO <sentencia>
```
Aquí <sentencia> se repite 10 veces

```
FOR V:= 1 TO -2
DO <sentencia>
```
Aquí <sentencia> se repite 0 veces



## Ejemplo

Escriba un programa para mostrar por pantalla todos los números enteros entre dos topes ingresados.

```
PROGRAM ListaNumeros;
{Muestra todos los números enteros desde tope inferior a tope superior}
VAR topeinf,topesup,num:INTEGER;
BEGIN
writeln('Ingrese los topes: ');
readln(topeinf, topesup);
writeln('Números');
FOR num:= topeinf TO topesup
    DO write(num,' , ');
END.
```

```
Ingrese los topes:
3 8
Números
3 , 4 , 5 , 6 , 7 , 8 ,
```

| topeinf | topesup | num |
|---|---|---|
| ? | ? | ? |
| 3 | 8 | 3 |
| | | 4 |
| | | 5 |
| | | 6 |
| | | 7 |
| | | 8 |



## Ejemplo

Escriba un programa para mostrar por pantalla todos los números enteros entre dos topes ingresados.

```
PROGRAM ListaNumeros;
{Muestra todos los números enteros desde tope inferior a tope superior}
VAR topeinf,topesup,num:INTEGER;
BEGIN
writeln('Ingrese los topes: ');
readln(topeinf, topesup);
writeln('Números');
FOR num:= topeinf TO topesup -1
    DO write(num,' , ');
write(topesup)
{Escribe por separado el último entero para evitar escribir la coma final}
END.
```

```
Ingrese los topes:
3 8
Números
3 , 4 , 5 , 6 , 7 , 8
```



Escriba un programa para mostrar por pantalla todos los números entre dos topes ingresados en cualquier orden.

```
PROGRAM ListaNumeros;
{muestra los números entre dos topes}
VAR topeinf,topesup,num,aux:INTEGER;
BEGIN
writeln('Ingrese topes: '); readln(topeinf, topesup);
IF topesup < topeinf {si los topes están invertidos}
THEN begin {intercambio los valores de los topes}
       aux:=topeinf;
       topeinf:=topesup;
       topesup:=aux;
     end;
writeln('Números');
FOR num:= topeinf TO topesup -1
    DO write(num,' , ');
write(topesup)
END.
```

```
Ingrese topes:
6 3
Números
3 , 4 , 5 , 6
```

| aux | topeinf | topesup | num |
|---|---|---|---|
| ? | ? | ? | ? |
| ? | 6 | 3 | ? |
| 6 | 3 | 6 | ? |
| | | | 3 |
| | | | 4 |
| | | | 5 |
| | | | 6 |

## Sentencia FOR-TO

**FOR V:= *exp1* TO *exp2* DO *sentencia***

- Tanto ***exp1*** **como** ***exp2*** pueden ser valores, variables, o expresiones siempre que sean del mismo tipo ordinal que V **(no puede ser tipo real).**
- Al comenzar a **V** se le asigna el valor de evaluar ***exp1*** y luego, **V** se incrementada automáticamente de a uno hasta llegar al valor de ***exp2***.

```
N:=2;
FOR V := 1+1 TO  N DO writeln(N);
FOR V := N TO  N+N DO  writeln(N);
FOR V := N div 2 TO (N+10) - N + 2 DO  writeln(N);
```



## FOR-TO con tipo CHAR

```
PROGRAM letras_y_codigos;
{muestra en pantalla los códigos
ASCII de algunas letras en columnas}
CONST  ultima='J'; columnas=3;
VAR  letra: char; contador: integer;
BEGIN
contador:=1;
FOR  letra:= 'A' TO ultima DO
 BEGIN {comienzo del ciclo for}
   write(letra,'=', ord(letra),'  ');
   {baja de renglón cada "columnas" veces}
   if contador mod columnas = 0
         then writeln;
   contador:=contador + 1;
 END {fin del ciclo FOR}
END.
```

```
A=65  B=66  C=67
D=68  E=69  F=70
G=71 H=72  I=73
J=74
```

Comienza con el valor 'A' y luego incrementa automáticamente de a uno pasando por todos los valores del código ASCII hasta llegar al valor final ('J')



## Problema Propuesto

- **Escriba un programa que calcule el promedio de una cantidad conocida (e ingresada por el usuario) de números reales.**

```
Ingrese la cantidad de valores: 4
Ingrese un valor: 8.2
Ingrese un valor: 0.2
Ingrese un valor: -3.0
Ingrese un valor: 5.2
El promedio es: 2.65
```



## Problema propuesto

Escriba un programa que calcule el promedio de una cantidad conocida (e ingresada por el usuario) de números reales.

Solución: El promedio consiste de sumar todo los números reales ingresados y dividir por la cantidad ingresada.

La dificultad que presenta no conocer hasta el momento de la ejecución cuantos valores hay que sumar, se resuelve calculando la suma a medida que los valores son ingresados (sin guardar los valores individuales)

```
Algoritmo "promedio":
   Leer cantidad
   Suma ← 0
   Repetir cantidad veces
      leer valor
      suma ← valor + suma
   Mostrar suma/cantidad
```

La flecha "←" es el símbolo de asignación usualmente usado en algoritmos.

El dato "suma" (inicialmente en cero) va acumulando el resultado de sumar los valores leídos.



## Un programa possible para promedio

```
PROGRAM Promedio;
{Calcula el promedio de una cantidad fija de valores
reales ingresados por el usuario }
VAR cantidad, control: integer;
suma, valor, prom: real;
BEGIN
writeln('Cantidad de valores: ');readln(cantidad);
suma:=0;  // valor inicial (neutro para la suma)
FOR  control:= 1  TO  cantidad DO
  begin
   writeln('Ingrese un valor: ');
   readln(valor);  // cada valor leído borra el anterior
   suma:=suma+valor; // pero la suma se acumula
  end;
prom:= suma/cantidad;
writeln('El promedio es: ',prom:0:2);
END.
```

| cantidad | suma | valor | control | prom |
|---|---|---|---|---|
| ? | ? | ? | ? | ? |
| 3 | 0 | ? | ? | ? |
| | 8 | 8 | 1 | ? |
| | 15 | 7 | 2 | ? |
| | 24 | 9 | 3 | 8 |
| | | | | |



## Problema propuesto

- **Escriba un programa para calcular un número natural elevado a una potencia (también natural).**
- **Ejemplos: $2^3=8$ $3^2=9$ $1^6=1$ $2^{10}=1024$**
  **$2^3=2*2*2=8$ $3^2=3*3=9$ $1^6=1*1*1*1*1*1=1$**
- **Solución: multiplicar "base" "exponente" veces**

```
Algoritmo "potencia":
   Leer base y exponente
   Potencia ← 1
   Repetir exponente veces
      potencia ← potencia * base
   Mostrar potencia
```

## Un posible programa para "potencia"

```
PROGRAM CalculoPotencia;
{Calcula un número natural elevado a una potencia (también natural) }
VAR base, exponente, potencia, aux: integer;
BEGIN
writeln('Ingrese base y exp : ');
readln(base, exponente);
potencia := 1;
FOR aux := 1 TO exponente
   DO potencia:=potencia * base;
write(base, ' a la ', exponente);
writeln(' es ',  potencia);
END.
```

| Base | exponente | aux | potencia |
|---|---|---|---|
| ? | ? | ? | ? |
| 2 | 7 | 1 | 2 |
| | | 2 | 4 |
| | | 3 | 8 |
| | | 4 | 16 |
| | | 5 | 32 |
| | | 6 | 64 |
| | | 7 | 128 |

```
Ingrese base y exp:
2  7
2 a la 7 es 128
```



## Problema propuesto

**Factorial de un número N se denota con N!**
**y se define: N! = 1 * 2 * 3 * 4 * ..... * N**
**0! = 1**

**Ejemplos:**
**1! = 1**
**2! = 2  3! = 6  4! = 24  5! = 120**
**6! = 720**
**7! = 5.040**
**8! = 40.320**
**9! = 362.880**
**10! = 3.628.800**

**Escriba un programa para calcular el factorial de un número ingresado por el usuario.**



## Algoritmo propuesto

**Problema: Escriba un programa para calcular el Factorial de un número N ingresado por el usuario.**

**Como N! = 1 * 2 * 3 * 4 * ..... * N**
**Por lo tanto tengo que hacer N multiplicaciones**

**Algoritmo factorial:**
- **Leer N**
- **factorial ← 1**
- **factor ← 1**
- **repetir N veces:**
  - **factorial ← factorial * factor**
  - **factor ← factor + 1**
- **Mostrar factorial en pantalla**



## Programa para "factorial"

```
PROGRAM   CalcularFactorial;
  { calcula factorial de un número ingresado}
VAR  numero, factor, factorial: INTEGER;
BEGIN
writeln('ingrese número >= 0');
readln(numero);
factorial:=1;
FOR factor:=1 TO numero
   DO  factorial:=factorial * factor;
Writeln(' El factorial de ',numero, ' es ' , factorial);
END.
```

| numero | factor | factorial |
|---|---|---|
| ? | ? | ? |
| 5 | 1 | 1 |
| | 2 | 2 |
| | 3 | 6 |
| | 4 | 24 |
| | 5 | 120 |
| | | |
| | | |



## Problemas para practicar

- **Escriba un programa que dados tres valores enteros V1, V2 y N ingresados por el usuario, muestre y cuente cuantos enteros hay entre V1 y V2 que sean múltiplos de N.**

```
Ingrese dos valores enteros: 7 30
Ingrese un divisor: 6
Entre 7 y 30, son múltiplos de 6:
12, 18, 24, 30
En total son 4 múltiplos.
```



## Continuará