Resolución de Problemas y Algoritmos

Clase 4
Estructura de control condicional.

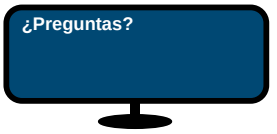
John von Neumann

Dr. Diego R. García

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

Conceptos de las clases anteriores

- **Algoritmo. Primitiva. Traza.**
- **Lenguaje de programación. Código fuente. Compilador.**
- **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.**



Objetivos de la materia RPA

El **objetivo** principal de RPA es que **los alumnos adquieran la capacidad de desarrollar programas de computadoras** para resolver problemas de pequeña escala.

El desarrollo de un programa se concibe como un proceso que abarca **varias etapas**:

1. La **interpretación** adecuada del enunciado a través del cual se plantea el problema.
2. El **diseño** de un **algoritmo** que especifica la resolución del problema.
3. La **implementación** del algoritmo en un lenguaje de programación imperativo.
4. La **verificación** de la solución.

Etapas

El objetivo principal de RPA es que los alumnos adquieran la capacidad de desarrollar programas de computadoras para resolver problemas de pequeña escala

El desarrollo de un programa se concibe como un proceso que abarca varias etapas:

1. La **interpretación** adecuada del enunciado a través del cual se plantea el problema.
2. El **diseño** de un **algoritmo** que es
3. La **implementación** del algoritmo
4. La **verificación** de la solución.

Estas etapas están en sintonía con el proceso de ingeniería de software que veremos más adelante

Concepto: lenguaje de programación

Un **lenguaje de programación** es un lenguaje artificial creado para expresar procesos que pueden ser llevados a cabo por computadoras. (Ejemplos: Pascal, C, C++, Java, PHP, Pearl, Smalltalk, Prolog, Lisp)

Un lenguaje de programación está definido por:

1. **un conjunto de símbolos**,
2. **reglas sintácticas** que definen su estructura, y
3. **reglas semánticas** que definen el significado de sus elementos.

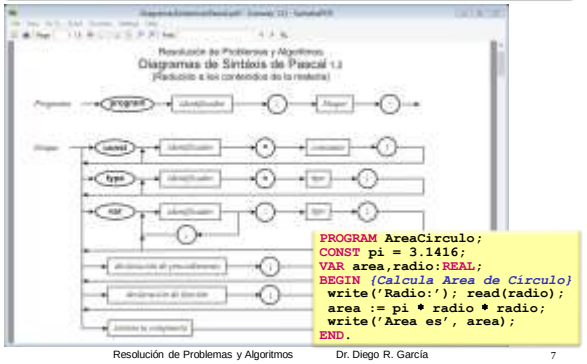
Conceptos: Diagrama Sintáctico

- La **sintaxis** de un lenguaje de programación es un conjunto de reglas que indica la estructura de los programas en ese lenguaje.

Un **diagrama sintáctico** (syntax diagram or railroad diagrams) es una forma gráfica de representar la sintaxis de un lenguaje de programación. Permite **describir sin ambigüedad** la sintaxis de un lenguaje de una manera simple y formal.

- Los **diagramas sintácticos** de Pascal que usaremos en RPA se encuentran disponibles en la página web de la materia.
- La **sintaxis** original escrita por N. Wirth está en la pág. 47 de : http://e-collection.library.ethz.ch/eserv/eth:3059/eth-3059-01.pdf

Parte del archivo con los diagramas sintácticos



Elementos de un diagrama sintáctico

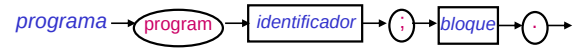
nombre → (1) **Un nombre y flecha** indican el comienzo de un diagrama para la definición de nombre.

texto — (2) **Las figuras "redondeadas"** indican que texto se debe incluir tal cual como aparece.

nombre — (3) Los **rectángulos** indican que nombre está definido en algún otro diagrama sintáctico.

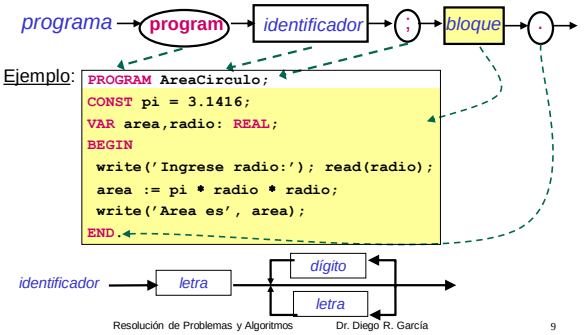
→ (4) Las **flechas** indican el orden de lectura en el diagrama.

Todos los programas en Pascal tienen esta estructura sintáctica:



Diagramas sintácticos

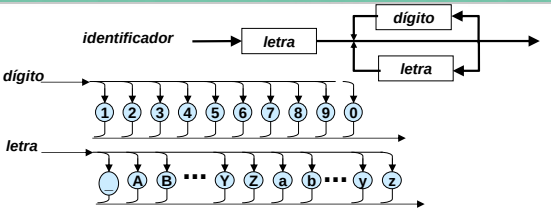
Todos los programas en Pascal tienen esta estructura sintáctica:



Ejemplo:

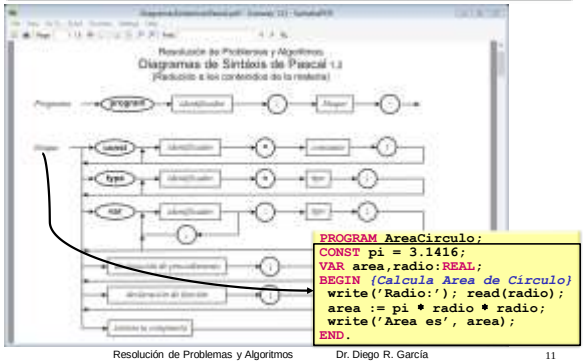
```
PROGRAM AreaCirculo;
CONST pi = 3.1416;
VAR area,radio: REAL;
BEGIN
 write('Ingrese radio:'); read(radio);
 area := pi * radio * radio;
 write('Area es', area);
END.
```


Diagramas sintácticos para "identificador"



- En letra se puede utilizar el símbolo "_" (underscore), las mayúsculas: ABCDEFGHIJKLMNOPQRSTUVWXYZ y las minúsculas: abcdefghijklmnopqrstuvwxyz
- No pueden utilizarse por ejemplo: á, ó, Ú, ñ, Ñ, -.

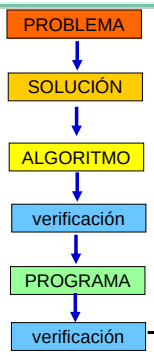
Parte del archivo con los diagramas sintácticos



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.**

Metodología general propuesta



Observación importante: los diagramas sintácticos nos ayudan a entender la sintaxis del lenguaje y poder identificar *errores sintácticos*.

Un programa puede ser sintácticamente válido (cumple con todas las reglas de sintaxis) pero igualmente tener errores que no son sintácticos. Para poder identificar errores en la lógica del programa se puede utilizar una traza.

Conceptos: diferentes clases de errores en programas

- **Error de compilación:** es un error detectado por el compilador al momento que se está realizando la compilación de un código fuente, por eso también se llama *error en tiempo de compilación*.
- **Error lógico:** también llamado **error de programación** (*bug*), es un error en la lógica del algoritmo o programa el cual causa que no se resuelva correctamente la tarea que debe hacer el programa.

Un profesional debe lograr que sus programas no tengan errores.

Para ello,

- los programas *deben verificarse* (en Inglés *testing*) con los suficientes casos de prueba, y
- si se detecta un mal funcionamiento con algún caso de prueba, entonces se deben *buscar y eliminar* los *errores* (*debugging*).

Primitiva de asignación

En una asignación: variable := expresión

1) primero se evalúa la expresión de derecha y se obtiene un valor,
2) luego se modifica el valor de la variable, perdiéndose el valor anterior.

Un dato sin valor a la derecha de ":=" es un ERROR de programación

```
PROGRAM AsignaMAL;
CONST d = 0.8;
VAR a,b,c: REAL;
BEGIN
  write('...');
  read(c);
  a:= b * c ;
  write(a);
END.
```

MAL

b no tiene valor

Traza de los valores almacenados en memoria

a	b	c
?	?	?
?	?	10

Observe que el programa AsignaMAL es sintácticamente valido y aún así tiene un error.

Intercambiar los valores de las variables

En una asignación: variable := expresión

1) primero se evalúa la expresión de derecha y se obtiene un valor,
2) luego modifica el valor de variable, perdiéndose el valor anterior.

Problema: Intercambiar los valores de dos variables `a` y `b` (de tipo char) de forma tal que el valor de `a` quede en `b` y el de `b` quede en `a`.

¿Qué casos de prueba usaría?

Observe que IntercambiaMAL es sintácticamente valido y aún así tiene un error.

```
PROGRAM IntercambiaMAL;
VAR a, b: char;
  BEGIN
   write('Ingrese 2 caracteres');
   readln(a);readln(b);
   a:= b;
   b:= a;
   writeln(a,' ',b);
  END.
```

MAL

Intercambiar los valores de las variables

En una asignación: variable := expresión

1) primero se evalúa la expresión de derecha y se obtiene un valor,
2) luego modifica el valor de variable, perdiéndose el valor anterior.

Problema: Intercambiar los valores de dos variables `a` y `b` (de tipo char) de forma tal que el valor de `a` quede en `b` y el de `b` quede en `a`.

```
PROGRAM IntercambiaMAL;
VAR a, b: char;
  BEGIN
   write('Ingrese 2 carac
   readln(a);readln(b);
   a:= b;
   b:= a;
   writeln(a,' ',b);
  END.
```

MAL

Traza de valores en memoria

a	b
?	?
'$'	?
'$'	'!'
'!'	'!'
'!'	'!'

Observe que IntercambiaMAL es sintácticamente valido y aún así tiene un error.

Contenedores de elementos de cierto tipo

- **Las variables pueden pensarse como recipientes que pueden contener un cierto tipo de elemento.**
- **Por ejemplo un vaso es un recipiente pensado para contener elementos de tipo líquido.**
- **Si tengo un vaso con una gaseosa, para poder tener ese mismo vaso con chocolatada, debo sacar la gaseosa para poder colocar la chocolatada.**

Piense ahora como resuelve el siguiente problema de dos hermanos pequeños: por error la taza de Mateo tiene jugo y la de María chocolatada, pero los niños quieren tomar cada uno en su propia taza lo que tiene el otro. ¿Cómo hace para intercambiar los líquidos de taza?

Intercambiar los valores de las variables

En una asignación: variable := expresión
1) primero se evalúa la expresión de derecha y se obtiene un valor,
2) luego modifica el valor de variable, perdiéndose el valor anterior.

Problema: Intercambiar los valores de dos variables `a` y `b` (de tipo char) de forma tal que el valor de `a` quede en `b` y el de `b` quede en `a`.

```
PROGRAM IntercambiaBIEN;
VAR a, b, aux: char;
    BEGIN
    write('Ingrese 2 caracteres');
    readln(a);readln(b);
    aux:= a;
    a:= b;
    b:= aux;
    write(a,' ', b);
    END.
```

Preservo el valor de a en aux.

BIEN

¿Qué casos de prueba usaría?

Intercambiar los valores de las variables

En una asignación: variable := expresión
1) primero se evalúa la expresión de derecha y se obtiene un valor,
2) luego se modifica el valor de variable, perdiéndose el anterior.

Problema: Intercambiar los valores de dos variables `a` y `b` de forma tal que el valor de `a` quede en `b` y el de `b` quede en `a`.

BIEN

```
PROGRAM IntercambiaBIEN;
VAR a, b, aux: char;
BEGIN
    write('Ingrese 2 caract
    readln(a);readln(b);
    aux:= a;
    a:= b;
    b:= aux;
    write(a,' ',b);
END.
```

Preservo el valor de a en aux.

Traza de valores en memoria

a	b	aux
?	?	?
'J'	?	?
'J'	'O'	?
'J'	'O'	'J'
'O'	'O'	'J'
'O'	'J'	'J'

Primitiva de asignación

El valor del resultado de la expresión tiene que pertenecer al tipo de la variable que se quiere modificar.

```
PROGRAM AsignaMAL;
VAR n,p: INTEGER;
BEGIN
  n:= 5;
  p:= n/2;
END.
```

MAL

n/2 da un resultado REAL y p es de tipo INTEGER

Traza de valores en memoria

n	p
?	?
5	?
5	

Primitiva de asignación

El valor del resultado de la expresión tiene que pertenecer al tipo de la variable que se quiere modificar.

```
PROGRAM AsignaBien;
VAR n,p: INTEGER;
BEGIN
  n:= 5;
  p:= trunc(n/2);
END.
```

BIEN

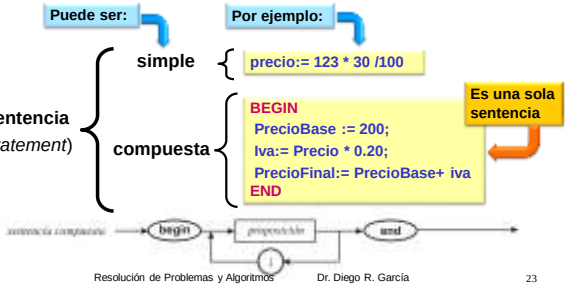
n/2 da un resultado REAL y p es de tipo INTEGER

Traza de valores en memoria

n	p
?	?
5	?
5	2

Sentencias en Pascal

Las sentencias *(statements)* en Pascal pueden ser simples o compuestas. En la jerga informática, la palabra ***statement*** se traduce a estas palabras del castellano: ***sentencia, instrucción*** o ***proposición***.



Sentencia compuesta en Pascal

Una sentencia compuesta comienza con **BEGIN** y termina con **END** **y permite definir una secuencia de sentencias como si fuera una única sentencia.**
Por ejemplo, la siguiente es una sentencia
(compuesta, a su vez, por una secuencia de 3 sentencias simples)

```
BEGIN
  PrecioBase := 200;
  Iva:= Precio * 0.20;
  PrecioFinal:= PrecioBase+ iva
END
```

El punto y coma es un separador de sentencias.

En la última sentencia de una secuencia el ";" no es necesario ya que no hay otra para separar de la última.

Uso del condicional: Problema simple

Problema: **Escriba un programa en Pascal para obtener el valor absoluto de un número.**

Solución:

Si el número es positivo o cero, el valor absoluto es el mismo número, de lo contrario es el número multiplicado por -1.

Algoritmo:

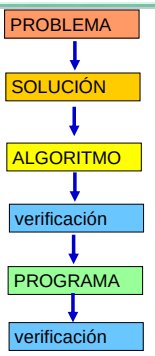
Leo Número
Si Número >= 0
entonces: val_abs es Número
de lo contrario: val_abs es Número * -1
Muestro val_abs en pantalla

Verificación:

Ejemplos significativos para casos de prueba:

un número positivo, uno negativo y cero, ejemplo: 3, 0 y -3

Metodología general propuesta



Veremos a continuación como codificar en el lenguaje Pascal una estructura condicional de la forma:

Si …condición…
entonces ….
de lo contrario …

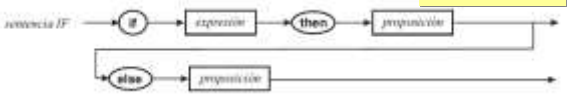
Estructura de control condicional (IF-THEN-ELSE)

IF expresión lógica (de tipo boolean)
THEN Una sentencia (simple o compuesta)
ELSE Una sentencia (simple o compuesta)

IF …
THEN …
ELSE …

IF …
THEN begin
…
…
end
ELSE begin
….
end

Sintaxis: ver el diagrama sintáctico.



Estructura de control condicional (IF-THEN-ELSE)

IF expresión lógica (de tipo boolean)
THEN Sentencia (simple o compuesta)
ELSE Sentencia (simple o compuesta)

False
True

IF …
THEN …
ELSE …

IF …
THEN begin
…
…
end
ELSE begin
….
end

Semántica:

- Si la evaluación de la **expresión lógica** da resultado **true**, entonces se ejecuta únicamente la sentencia que sigue al "**THEN**" (ya sea una sentencia simple o compuesta) .
- Si en cambio, la evaluación de la **expresión lógica** da **false**, entonces se ejecuta solamente la sentencia que sigue al "**ELSE**".

Programa para valor absoluto

```
program valor_absoluto;
var   numero, val_abs: real;
{Realiza el cálculo del valor absoluto de un número}
begin
Write ('Ingrese un número');
readln(numero);
IF numero >= 0
   THEN val_abs := numero
   ELSE val_abs := (-1) * numero;
writeln(' Su valor absoluto es: ', val_abs);
end.
```

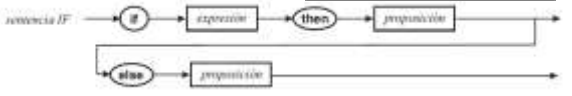
Realice trazas para los casos de prueba: 3, 0 y -3

Observe: no lleva ";" antes del ELSE

Estructura de control condicional (IF-THEN)

IF expresión lógica
THEN Sentencia 1 (simple o compuesta)

IF …
THEN begin
…
end



Sintaxis: vea en el diagrama sintáctico que en la sentencia IF no es obligatorio un ELSE (es opcional).
Semántica: Si la evaluación de la expresión lógica da resultado **true**, entonces se ejecuta solamente la sentencia que sigue al "**THEN**" (sea simple o compuesta) .
Si en cambio la evaluación de la expresión lógica da **false**, se sigue con la ejecución de la sentencias que siguen al **IF** (si es que existen).

Otro programa para valor absoluto

Otra solución sin usar ELSE

```
program valor_absoluto;
var   numero, val_abs: real;
{Realiza el cálculo del valor absoluto de un número}
begin
Write ('Ingrese un número'); readln(numero);
val_abs:= numero;
IF numero < 0
   THEN val_abs := (-1) * numero;
writeln(' Su valor absoluto es: ', val_abs);
end.
```

Realice trazas para los casos de prueba: 3, 0 y -3

Problema simple propuesto

Problema: Considerando únicamente las letras a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z; escriba un programa que lea un caracter y distinga si se trata de una letra mayúscula o minúscula. Obs: para simplificar no incluimos las vocales con acentos ni la letra Ñ, pero lo haremos más adelante.

Solución: un caracter ASCII entre 'A' y 'Z' es una letra mayúscula, un caracter entre 'a' y 'z' es una letra minúscula.

Algoritmo:

- **leer el caracter**
- **Si está entre 'A' y 'Z' entonces es una mayúscula**
- **Si está entre 'a' y 'z' entonces es una minúscula**

Verificación:

casos de prueba: una mayúscula, una minúscula y un carácter que no sea una letra (ejemplos: 'G', 'g', '3' , '$')

No cantemos victoria antes de gloria…

```
PROGRAM Veamos;
var   ch: char;
{Este programa intenta distinguir mayúsculas y minúsculas}
BEGIN
write('Ingrese un caracter:');
readln(ch);
IF  (ch >= 'A') and (ch <= 'Z')
   THEN writeln(ch, ' es una mayúscula.')
   ELSE writeln(ch, ' es una minúscula.');
END.
```

Realice trazas para los casos de prueba: 'G', 'g', '3' y '$'

La traza para '3' y para '$' muestra que hay un ERROR. Usando ELSE, cualquier CHAR que no sea mayúscula se considera minúscula, lo cual es incorrecto.

Un programa correcto

```
program mayucula_o_minuscula;
var   ch: char;
{Este programa permite distinguir mayúsculas y minúsculas}
begin
write('Ingrese un caracter:');
readln(ch);
IF  (ch >= 'A') and (ch <= 'Z')
   then writeln(ch, ' es una mayúscula.');
IF  (ch >= 'a') and (ch <= 'z')
   then writeln(ch, ' es una minúscula.');
end.
```

Realice trazas para los casos de prueba: 'G', 'g', '3' y '$'

Encuentre el error…

```
program EncuentreError;
var   ch: char;
{Este programa tiene un error}
begin
write('Ingrese un caracter:');
readln(ch);
IF  (ch >= 'A') and (ch <= 'Z')
   THEN writeln(ch, ' es una mayúscula.');
IF  (ch >= 'a') and (ch <= 'z')
   THEN writeln(ch, ' es una minúscula.')
   ELSE writeln(ch, 'no es una letra');
end.
```


Continuará …

Información adicional

Sobre como dos personas que comparten (¿de casualidad?) un curso en una universidad, y luego, otras dos que coinciden (¿de casualidad?) en una estación de tren, cambian el rumbo de la computación moderna.

En 1941 el físico John Mauchly, viajó para hacer un curso en la Universidad de Pennsylvania y allí conoce al ing. electrónico John Eckert. …

Primeras computadoras: ENIAC & EDVAC

En 1943 John Mauchly, y John Eckert diseñaran ENIAC (Electronic Numerical Integrator And Computer).

La programación era por hardware y reprogramarla costaba días. Esta titánica tarea estaba a cargo de 6 mujeres con grandes habilidades matemáticas y lógicas, que iban inventando la programación a medida que la realizaban:

Betty Snyder Holberton, Jean Jennings Bartik, Kathleen McNulty Mauchly Antonelli, Marlyn Wescoff Meltzer, Ruth Lichterman Teitelbaum y Frances Bilas Spence

Computadora ENIAC (1946) (http://es.wikipedia.org/wiki/ENIAC)

Primeras computadoras: ENIAC & EDVAC

Eckert y Mauchly conscientes de las limitaciones de ENIAC empezaron con un nuevo diseño para una nueva computadora. La cual se llamó EDVAC (Electronic Discrete Variable Automatic Computer).
Eckert propuso una memoria de mercurio para guardar tanto el programa como datos.
El matemático Goldstine también era parte del proyecto EDVAC, y en 1945, en una charla casual (en una espera en una estación de tren) entusiasmó al matemático húngaro John von Neumann a formar parte del proyecto.
La arquitectura de EDVAC incorporó el concepto de programa almacenado en memoria y la codificación en binario en lugar de decimal.

Computadora EDVAC (1945)
(http://es.wikipedia.org/wiki/EDVAC)

El origen del término arquitectura von Neumann

John von Neumann fue una persona muy inteligente. Trabajó en: matemática, lógica, programación, energía atómica, física y computación teórica. [ver más]

John von Neumann

Gracias al encuentro casual con Goldstine en 1945, von Neumann se reunió con Mauchly y Eckert, y esa reunión cambiaría la historia…

Luego mientras von Neumann regresaba en tren en a su casa, escribió un reporte (a mano) de todo lo que hablaron en la reunión.

Al llegar, su secretaria tipió el reporte con máquina de escribir y se lo envió a Goldstine, quien a su vez lo distribuyó (por carta) entre sus colegas de todo el mundo, quedando como autoría solo el nombre de John Von Neumann (sin incluir a Mauchly y Eckert). ☹

Así surgió el término "arquitectura de von Neumann" para denominar las nuevas ideas que tenía EDVAC y que aún perduran en las computadoras actuales.