

Introducción a la Algorítmica y Programación (3300)

Prof. Ariel Ferreira Szpiniak - afferreira@exa.unrc.edu.ar
Departamento de Computación
Facultad de Cs. Exactas, Fco-Qcas y Naturales
Universidad Nacional de Río Cuarto

Teoría 11

Diseño: notaciones, método descendente.

Programación estructurada



2017 Lic. Ariel Ferreira Szpiniak 1

¿Cómo se Representa un Algoritmo?

La representación más obvia es mediante código fuente en algún lenguaje de programación (Pascal, C, Java, etc).

Esto no es recomendable porque es muy propenso a errores. Antes hay que entender lo que hay que solucionar y esbozar una solución en un lenguaje más general.

Para ello existen diversas técnicas o notaciones.



2017 Lic. Ariel Ferreira Szpiniak 2

Notaciones para el diseño de algoritmos

• Notación algorítmica o Pseudocódigo

Serie de normas léxicas y gramaticales parecidas a la mayoría de los lenguajes de programación pero de un nivel más general.

• Diagramas de flujo o Flowcharts

Esquema para representar gráficamente un algoritmo basado en diferentes símbolos. Representación visual del flujo de control de un algoritmo.



2017 Lic. Ariel Ferreira Szpiniak 3

Notaciones para el diseño de algoritmos

Notación algorítmica o Pseudocódigo (falso lenguaje)

- Forma de expresión intermedia entre el lenguaje natural y los lenguajes de programación, pero con mayor poder expresivo que éstos últimos.
- Permiten codificar un programa con mayor agilidad y con la misma validez semántica.
- Se utilizan fundamentalmente en la fase de diseño.
- No hay ningún compilador o intérprete, salvo el diseñador.
- Su objetivo es permitir que el programador se centre en los aspectos lógicos de la solución a un problema.

Pseudocódigo = Pseudo (Supuesto) + Código (Instrucción)



2017 Lic. Ariel Ferreira Szpiniak 4

Notaciones para el diseño de algoritmos

Diagrama de flujo

- Son esquemas para representar gráficamente un algoritmo.
- Se utilizan también en economía y procesos industriales.
- Utilizan una serie de símbolos con significados especiales para representar operaciones específicas.
- Se los llama diagramas de flujo porque los símbolos utilizados se conectan por medio de flechas para indicar la secuencia de operación.
- Se utilizaron ampliamente en programación, tanto estructurada como no estructurada (uso de **go to**). En la actualidad también se utilizan.
- Muchos métodos de análisis y diseño se basan en esta idea (DFD, BPMN - DPN, DTE, DA, Workflow, etc.).



2017 Lic. Ariel Ferreira Szpiniak 5

Notaciones para el diseño de algoritmos

Diagrama de flujo - Símbolos más utilizados



Rectángulo: representa un evento o proceso determinado. Rectángulo redondeado: representa un evento que ocurre de forma automática.



Rombo: representa una condición con dos caminos posibles.



Flecha: indica el sentido y trayectoria del proceso de información o tarea.



Círculo: representa un punto de conexión entre procesos.

Existe una variedad de formas para denotar las entradas, salidas, etc.



2017 Lic. Ariel Ferreira Szpiniak 6

Notaciones para el diseño de algoritmos

Diagrama de flujo – Ventajas y Desventajas

Ventajas

- Simplicidad
- Habilidad para representar visualmente el flujo de control.

Desventajas

- Los flowcharts altamente detallados pueden generar errores o imprecisiones.
- Se requiere tiempo extra para realizar los diagramas
- No son tan útiles en un contexto de lenguajes de programación orientado a objetos.



2017 Lic. Ariel Ferreira Szpiniak 7

Notaciones para el diseño de algoritmos

Ventajas de Pseudocódigo sobre Diagramas de flujo

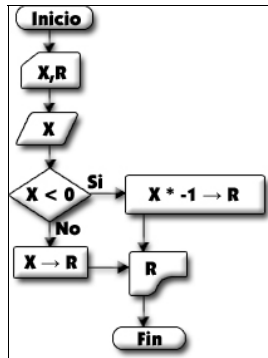
1. La programación estructurada hace intuitivo el flujo del programa (no hace falta el diagrama).
2. Permite representar de forma fácil operaciones repetitivas complejas.
3. Son más fáciles de entender que los diagramas de flujo.
4. Está más cerca de la programación. Es más sencilla la tarea de pasar de pseudocódigo a un lenguaje de programación.
5. Si se siguen las reglas de indentación se puede observar claramente los niveles en la estructura del programa.
6. Los diagramas se hacen inmanejables cuando la resolución del problema es larga.



2017 Lic. Ariel Ferreira Szpiniak 8

Notaciones para el diseño de algoritmos

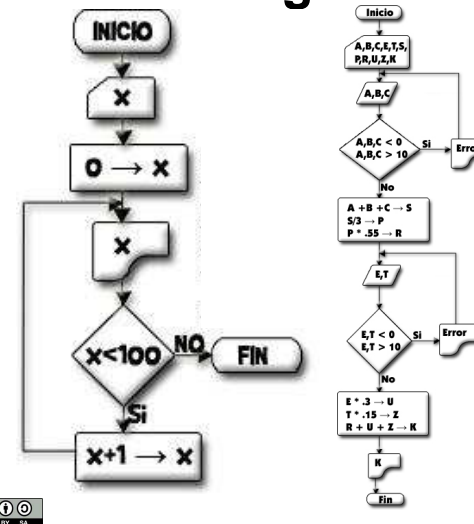
Problema: Leer un número y escribir el valor absoluto del mismo.



Algoritmo ValorAbsoluto
Lexico
 $x, r \in E$
Inicio
 Leer(x)
si $x < 0$
entonces {num es negativo}
 $r \leftarrow x * -1$
sino {num es positivo}
 $r \leftarrow x$
fsi
 Escribir(r)
Fin

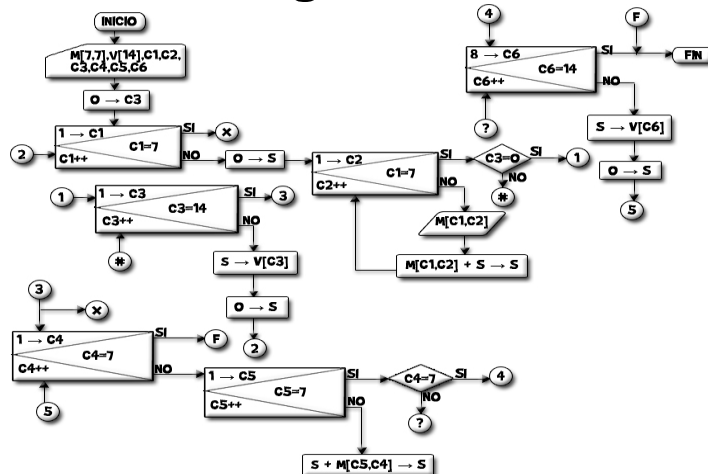


Notaciones para el diseño de algoritmos

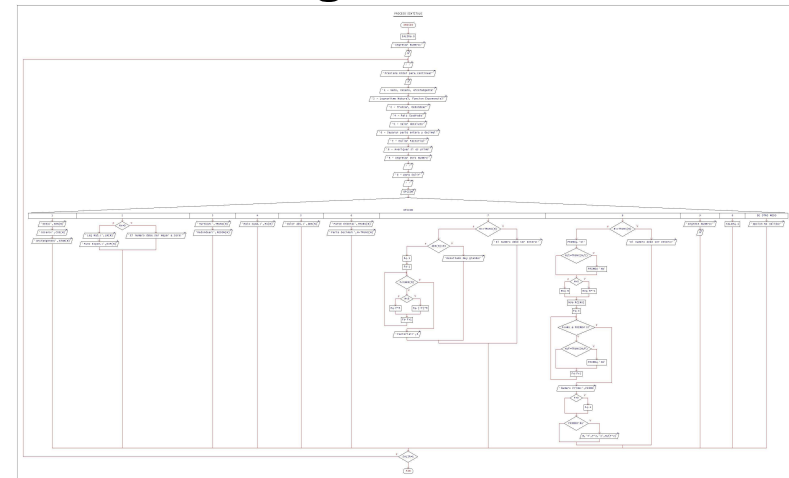


¿y si es mucho más largo?

Notaciones para el diseño de algoritmos






Notaciones para el diseño de algoritmos



Programación Estructurada

- La **Composición de Algoritmos** tiene su base en el concepto conocido como **Programación Estructurada**.
- Si bien la **programación** (codificación de un algoritmo) es una etapa posterior al **diseño** (construcción del algoritmo), los lineamientos propuestos pueden ser extrapolados a la etapa de diseño.
- Esto permite obtener un **diseño estructurado** que adquiere las mismas características que la **programación estructurada**.






Programación Estructurada

-  La programación estructurada **es un estilo de programación** que consiste en construir programas de fácil comprensión. Produce código con un **flujo limpio**, un **diseño claro** y un cierto grado de **modularidad** o de estructura jerárquica.
-  Entre los beneficios de la programación estructurada se encuentran la facilidad de **escritura**, **verificación** y **modificación** así como también la **legibilidad** por parte de otros programadores.
-  La programación estructurada permite la **reusabilidad** del código, extrayendo módulos que pueden ser utilizados en otros programas, con nulas o mínimas readaptaciones.



2017 Lic. Ariel Ferreira Szpiniak 14

Programación Estructurada

-  La programación estructurada se adapta naturalmente a la metodología de desarrollo descendente que plantea la división del problema en subproblemas de menor complejidad.
-  Una vez resueltos los subproblemas se procede a unificar las soluciones.
-  La programación estructurada propone la utilización de **tres estructuras básicas** de control lógico las cuales se conocen como: **secuencia**, **condición** e **iteración**.
-  Cualquier programa, no importa el tipo de trabajo que ejecute ni el nivel de complejidad, puede ser elaborado utilizando únicamente las tres estructuras básicas.
-  Estas estructuras se pueden componer una seguida de otras a anidarlas, es decir, “meter” una estructura dentro de otra.



2017 Lic. Ariel Ferreira Szpiniak 15

Programación Estructurada Teorema

El principio fundamental de la programación estructurada se basa en:

- a) diseño descendente del programa,
- b) estructuras de control limitadas,
- c) ámbito limitado de las estructuras de datos del programa.

Para realizar un programa estructurado existen solo tres tipos básicos de estructuras de control:

- **Secuencial:** ejecuta una sentencia detrás de otra.
- **Condicional:** evalúa una expresión y, dependiendo del resultado, se decide la siguiente sentencia a ejecutar.
- **Iterativa:** repite un bloque de sentencias hasta que sea verdadera una determinada condición.



2017 Lic. Ariel Ferreira Szpiniak 16

Programación Estructurada

Teorema

Teorema Fundamental de la programación estructurada

“Todo programa propio se puede escribir utilizando únicamente las estructuras de control secuencial, condicional e iterativa” [Böh66]

Un *programa propio* es aquel que:

- Tiene un único punto de entrada y un único punto de salida.
- Existen caminos desde la entrada hasta la salida que pasan por todas las partes del programa.
- Todas las instrucciones son ejecutables y no existen ciclos sin fin.

La utilización de la sentencia **GOTO** es totalmente innecesaria.

Un programa con GOTO es más difícil de entender y verificar.

[Böh66] C.Böhm, G.Jacopini, Comm. ACM vol.9, nº5, 366-371,1966



2017 Lic. Ariel Ferreira Szpiniak 17

Pascal

Un lenguaje utilizado para programación estructurada

- Desarrollado por el Niklaus Wirth (entre 1968 y 1970).
- Es un lenguaje de programación de alto nivel y de propósito general.
- El objetivo de Wirth fue crear un lenguaje que facilitara el aprendizaje de técnicas de programación estructurada.
- Se caracteriza por ser un lenguaje de programación estructurado fuertemente tipado:
 - El código está dividido en porciones fácilmente legibles.
 - El tipo de dato de todas las variables debe ser declarado previamente.

Algunas consideraciones importantes:

- Case: diferencias con el SEGUN
- Comentarios
- Punto y coma
- BEGIN ... END



2017 Lic. Ariel Ferreira Szpiniak 18

Método Descendente

Pasos para solucionar un problema

• Análisis

El problema debe ser claramente especificado y entendido.

• **Diseño**

Construcción de una solución general del problema.

• Implementación

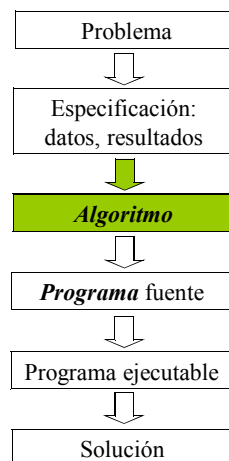
Traducción del algoritmo a un lenguaje de programación de alto nivel.

• Compilación

Escritura del programa Pascal, usando en nuestro caso el Compilador Turbo Pascal 7.0.

• Ejecución y Prueba

Corrida y prueba de funcionamiento del programa en la computadora.



2017 Lic. Ariel Ferreira Szpiniak 19

Método Descendente

Introducción

- Un enfoque para la resolución de un problema complejo es el de subdividirlo en problemas más simples (de menor complejidad) y luego resolver estos últimos.
- Este proceso a su vez puede ser repetido hasta que los problemas a resolver sean lo suficientemente simples.
- Esta técnica se conoce con el nombre de **divide and conquer** (*divide y vencerás*). El enfoque de obtener la solución principal a partir de soluciones de los subproblemas es llamado **método descendente o diseño descendente**.



2017 Lic. Ariel Ferreira Szpiniak 20

Método Descendente Introducción

- Las soluciones para problemas complejos diseñadas en forma descendente pueden ser resueltas mediante algoritmos.
- El problema principal es resuelto por el correspondiente **algoritmo principal**.
- Soluciones a los subproblemas pueden ser desarrolladas como subalgoritmos. Esto permite obtener algoritmos modulares. Estos subalgoritmos, acciones no primitivas, pueden ser **acciones** y **funciones**.



2017 Lic. Ariel Ferreira Szpiniak 21

Método Descendente Introducción

- Los algoritmos diseñados en forma descendente pueden ser fácilmente implementados en computadores usando un lenguaje estructurado como Pascal.
- El problema principal es resuelto por el correspondiente **programa principal**.
- Los subalgoritmos son implementados como subprogramas, que en Pascal corresponden a bloques de acciones o **acciones** (procedures) y **funciones** (functions).



2017 Lic. Ariel Ferreira Szpiniak 22

Método Descendente Ventajas

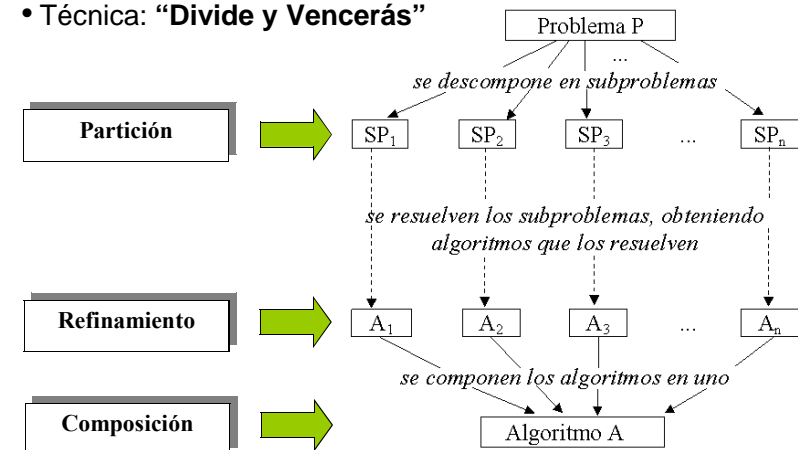
- Permite una mejor comprensión del problema.
- El problema puede ser lógicamente organizado en unidades conceptuales independientes que llamaremos **módulos**.
- De esta forma le estamos dando una **estructura** a la solución del problema.
- Modificaciones o cambios, abstracción y reutilización.
- Correctitud.



2017 Lic. Ariel Ferreira Szpiniak 23

Método Descendente Partición y Refinamiento

- Técnica: “Divide y Vencerás”



2017 Lic. Ariel Ferreira Szpiniak 24

Método Descendente

Forma de aplicarlo

• Análisis

Preguntas concernientes a la entrada: - ¿Cuáles y cuántos son los valores de entrada? - ¿Cuáles son valores válidos de entrada?
Preguntas concernientes a los resultados (salida): - ¿Cuáles y cuántos son los valores del resultado? - ¿Cuáles son valores válidos del resultado? - ¿Cómo se llega a esos resultados?
Conocimiento (qué debo saber)

• Diseño – primera capa

Partición: Se descompone el problema principal en subproblemas

Definimos para cada subproblema:

- Módulo a utilizar (acción o función).
- Especificación (más adelante veremos como especificar mejor).

• Diseño – segunda capa

Refinamiento: Se resuelven los subproblemas obteniendo los subalgoritmos.

- Nombre del módulo. Parámetros: tipo de cada uno y tipo de pasaje (dato, dato-resultado, resultado).
- Estructuras iterativas necesarias y sus características: cantidad de iteraciones (determinada o no), cantidad mínima de iteraciones (0, 1), etc.

• Diseño – tercera capa

Composición: Se componen los subalgoritmos en un algoritmo solo.

Definimos para el problema en general:

- Léxico: tipos que utilizaremos, nombre de cada variable con su tipo.
- Estructuras iterativas necesarias y sus características: cantidad de iteraciones (determinada o no), cantidad mínima de iteraciones (0, 1), etc.

• Implementación

Traducción del algoritmo a un lenguaje de programación de alto nivel (Pascal o C en nuestro caso).



2017 Lic. Ariel Ferreira Szpiniak 25

Método Descendente

Ejemplo

Desarrollar un algoritmo que solicite al usuario la base y altura de un triángulo, calcule, almacene y muestre el área del mismo.

• Análisis

Preguntas concernientes a la entrada:

- ¿Cuáles y cuántos son los valores de entrada? - ¿Cuáles son valores válidos de entrada?

Preguntas concernientes a los resultados (salida):

- ¿Cuáles y cuántos son los valores del resultado? - ¿Cuáles son valores válidos del resultado?
- ¿Cómo se llega a esos resultados?

Relación entre datos de Entrada y Resultados



2017 Lic. Ariel Ferreira Szpiniak 26

Método Descendente

Ejemplo (cont.)

• Análisis

- **Datos:** son necesarias la base y altura del triángulo. Ambos son números reales positivos.
- **Resultado:** el área del triángulo que es un número real positivo.
- **Conocimiento:** el área de un triángulo se calcula de la siguiente manera $a = (b * h) / 2$



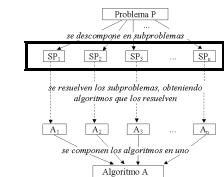
2017 Lic. Ariel Ferreira Szpiniak 27

Método Descendente

Ejemplo (cont.)

• Diseño – primera capa

Partición



Se descompone el problema principal en subproblemas:

SP1: lectura de datos

SP2: cálculo del área

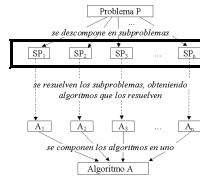
SP3: mostrar área



2017 Lic. Ariel Ferreira Szpiniak 28

Método Descendente Ejemplo (cont.)

- Diseño – primera capa
Partición



Definimos para cada subproblema:

- Módulo a utilizar (acción o función).
- Especificación (precondición, y poscondición o definición).



Método Descendente Ejemplo (cont.)

- Diseño – primera capa

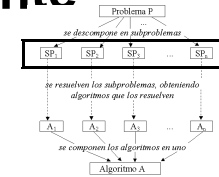
Definimos para cada subproblema:

SP1: lectura de datos

- Módulo a utilizar: acción

- {Pre-condición: ninguna}

- {Pos-condición: base y altura deben tener un valor mayor que 0}



Método Descendente Ejemplo (cont.)

- Diseño – primera capa

Definimos para cada subproblema:

SP2: cálculo del área

- Módulo a utilizar: acción*

- {Pre-condición: base y altura deben tener un valor mayor que 0}

- {Pos-condición: área debe ser igual a $(base * altura) / 2$ }

* También puede ser una función



Método Descendente Ejemplo (cont.)

- Diseño – primera capa

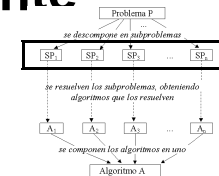
Definimos para cada subproblema:

SP3: mostrar área

- Módulo a utilizar: acción

- {Pre-condición: área debe ser igual a $(base * altura) / 2$ }

- {Pos-condición: mostrar el valor de área por pantalla}



Método Descendente Ejemplo (cont.)

• Diseño – segunda capa

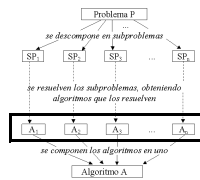
Refinamiento

Se resuelven los subproblemas obteniendo los subalgoritmos:

- Nombre del módulo: LeerDatos
- Parámetros: $b \in \mathbb{R}$, $h \in \mathbb{R}$ de tipo resultado
- Estructuras iterativas: una estructura para validar que b sea positiva y otra para h . Una iteración como mínimo de cada una para tomar el valor de cada variable.

```

Acción LeerDatos(resultado  $b \in \mathbb{R}$ ,  $h \in \mathbb{R}$ )
Inicio
  Repetir
    Leer(b)
  Hasta que  $b > 0$ 
  Repetir
    Leer(h)
  Hasta que  $h > 0$ 
Fin
    
```



Método Descendente Ejemplo (cont.)

• Diseño – segunda capa

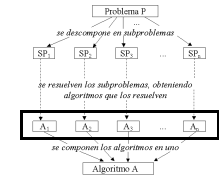
Refinamiento

Se resuelven los subproblemas obteniendo los subalgoritmos:

- Nombre del módulo: CalcArea
- Parámetros: $b \in \mathbb{R}$, $h \in \mathbb{R}$ de tipo dato, $a \in \mathbb{R}$ de tipo resultado
- Estructuras iterativas: no son necesarias.

```

Acción CalcArea(dato  $b \in \mathbb{R}$ ,  $h \in \mathbb{R}$ , resultado  $a \in \mathbb{R}$ )
Inicio
   $a \leftarrow (b * h) / 2$ 
Fin
    
```



Método Descendente Ejemplo (cont.)

• Diseño – segunda capa

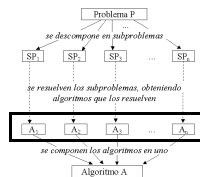
Refinamiento

Se resuelven los subproblemas obteniendo los subalgoritmos:

- Nombre del módulo: MostrarArea
- Parámetros: $a \in \mathbb{R}$ de tipo dato
- Estructuras iterativas: no son necesarias.

```

Acción MostrarArea(dato  $a \in \mathbb{R}$ )
Inicio
  Escribir(a)
Fin
    
```



Método Descendente Ejemplo (cont.)

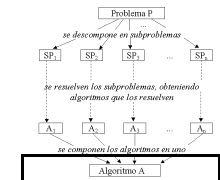
• Diseño – tercera capa

Composición

Se componen los subalgoritmos en un algoritmo solo.

Definimos para el problema en general:

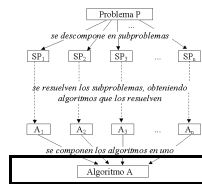
- Léxico: tipos que utilizaremos, nombre de cada variable con su tipo.
- Estructuras iterativas necesarias y sus características: cantidad de iteraciones (determinada o no), cantidad mínima de iteraciones (0, 1), etc.



Método Descendente Ejemplo (cont.)

• Diseño – tercera capa Composición

Se componen los subalgoritmos en un algoritmo solo.



Definimos para el problema en general:

• Léxico:

base, altura $\in \mathbb{R}$ {base y altura del triángulo}
area $\in \mathbb{R}$ {área del triángulo}

• Estructuras iterativas: no son necesarias.

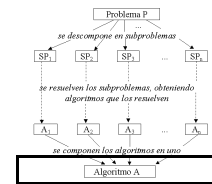


2017 Lic. Ariel Ferreira Szpiniak 37

Método Descendente Ejemplo (cont.)

• Diseño – tercera capa Composición

Se componen los subalgoritmos en un algoritmo solo.



Algoritmo CalcularAreaTriángulo

Lexico

base, altura $\in \mathbb{R}$ {base y altura del triángulo}
area $\in \mathbb{R}$ {área del triángulo}

Acción LeerDatos(resultado b $\in \mathbb{R}$, h $\in \mathbb{R}$)

Inicio

Repetir

Leer(b)

Hasta que b > 0

Repetir

Leer(h)

Hasta que h > 0

Fin {continúa...}



2017 Lic. Ariel Ferreira Szpiniak 38

Método Descendente Ejemplo (cont.)

• Diseño – tercera capa

Acción CalcArea(dato b $\in \mathbb{R}$, h $\in \mathbb{R}$, result

Inicio

a \leftarrow (b * h) / 2

Fin

Acción MostrarArea(dato a $\in \mathbb{R}$)

Inicio

Escribir(a)

Fin

Inicio

LeerDatos(base, altura)

CalcArea(base, altura, area)

MostrarArea(area)

Fin



2017 Lic. Ariel Ferreira Szpiniak 39

Método Descendente Pasos para solucionar un problema

• Análisis

El problema debe ser claramente especificado y entendido.

• Diseño

Construcción de una solución general del problema.

• Implementación

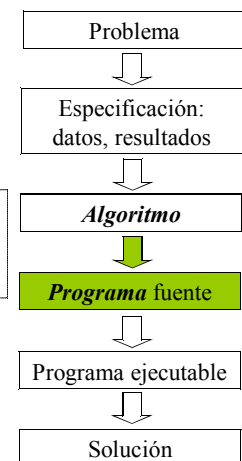
Traducción del algoritmo a un lenguaje de programación de alto nivel.

• Compilación

Escritura del programa Pascal, usando en nuestro caso el Compilador Turbo Pascal 7.0.

• Ejecución y Prueba

Corrida y prueba de funcionamiento del programa en la computadora.



2017 Lic. Ariel Ferreira Szpiniak 40

Método Descendente Ejemplo (cont.)

• Implementación

Traducción del algoritmo a un lenguaje de programación de alto nivel (Pascal en nuestro caso).

```
PROGRAM CalcularAreaTriangulo;
USES CRT;
VAR
  base, altura: Real;      {base y altura del triángulo}
  area: Real;              {área del triángulo}
PROCEDURE LeerDatos(VAR b:Real; VAR h: Real);
BEGIN
  REPEAT
    READ(b)
  UNTIL b>0;
  REPEAT
    READ(h)
  UNTIL h>0
END;
```



2017 Lic. Ariel Ferreira Szpiniak 41

Método Descendente Ejemplo (cont.)

• Implementación

Traducción del algoritmo a un lenguaje de programación de alto nivel (Pascal en nuestro caso).

```
PROCEDURE CalcArea(b:Real; h: Real; VAR a:Real);
BEGIN
  a := (b * h)/2
END;
PROCEDURE MostrarArea(a:Real);
BEGIN
  WRITELN(a:5:2)
END;
BEGIN
  LeerDatos(base, altura);
  CalcArea(base, altura, area);
  MostrarArea(area);
  READKEY
END.
```



2017 Lic. Ariel Ferreira Szpiniak 42

Método Descendente Ejemplo (cont.)

¿Qué pasaría si decidiera resolver el SP2 con una función?

• Diseño – primera capa

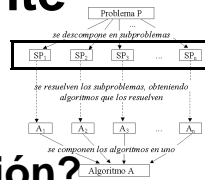
Definimos para cada subproblema:

SP2: cálculo del área

• Módulo a utilizar: función

• {Pre-condición: base y altura deben tener un valor mayor que 0}

• {Definición: área debe ser igual a $(base * altura) / 2$ }



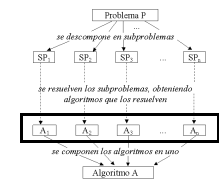
2017 Lic. Ariel Ferreira Szpiniak 43

Método Descendente Ejemplo (cont.)

• Diseño – segunda capa Refinamiento

Se resuelven los subproblemas obteniendo los subalgoritmos:

- Nombre del módulo: CalcArea
- Parámetros: $b \in R$, $h \in R$ de tipo dato. Devuelve un valor R
- Estructuras iterativas: no son necesarias.



```
Función CalcArea(dato  $b \in R$ ,  $h \in R$ )  $\rightarrow R$ 
Inicio
   $\leftarrow (b * h)/2$ 
Fin
```



2017 Lic. Ariel Ferreira Szpiniak 44

Método Descendente Ejemplo (cont.)

• Diseño – tercera capa Composición

Se componen los subalgoritmos en un algoritmo solo.

Algoritmo CalcularAreaTriángulo

Lexico

base, altura $\in \mathbb{R}$ {base y altura del triángulo}

area $\in \mathbb{R}$ {área del triángulo}

Acción LeerDatos(resultado b $\in \mathbb{R}$, h $\in \mathbb{R}$)

Inicio

Repetir

Leer(b)

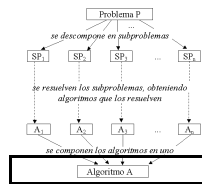
Hasta que b > 0

Repetir

Leer(h)

Hasta que h > 0

Fin {continúa...}



2017 Lic. Ariel Ferreira Szpiniak 45

Método Descendente Ejemplo (cont.)

• Diseño – tercera capa

Función CalcArea(dato b $\in \mathbb{R}$, h $\in \mathbb{R}$) $\rightarrow \mathbb{R}$

Inicio

$\leftarrow (b * h) / 2$

Fin

Acción MostrarArea(dato a $\in \mathbb{R}$)

Inicio

Escribir(a)

Fin

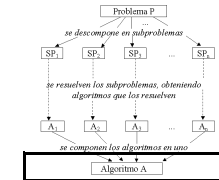
Inicio

LeerDatos(base, altura)

area \leftarrow CalcArea(base, altura)

MostrarArea(area)

Fin



2017 Lic. Ariel Ferreira Szpiniak 46

Método Descendente Ejemplo

Variante del problema anterior:

Desarrollar un algoritmo que solicite al usuario, **tantas veces como lo desee**, la base y altura de un triángulo, calcule, almacene y muestre el área del mismo.



2017 Lic. Ariel Ferreira Szpiniak 47

Método Descendente Ejemplo (cont.)

• Análisis

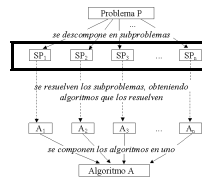
- **Datos:** son necesarias la base y altura del triángulo. Ambos son números reales positivos.
- **Resultado:** el área del triángulo que es un número real positivo.
- **Adicionales** (conocimiento): el área de un triángulo se calcula de la siguiente manera $a = (b * h) / 2$



2017 Lic. Ariel Ferreira Szpiniak 48

Método Descendente Ejemplo (cont.)

• Diseño – primera capa Partición



Se descompone el problema principal en subproblemas:

SP1: lectura de datos

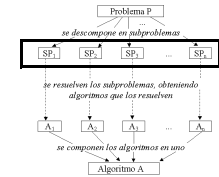
SP2: cálculo del área

SP3: mostrar área



Método Descendente Ejemplo (cont.)

• Diseño – primera capa Partición



Definimos para cada subproblema:

- Módulo a utilizar (acción o función).
- Especificación.



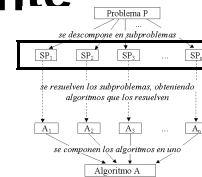
Método Descendente Ejemplo (cont.)

• Diseño – primera capa

Definimos para cada subproblema:

SP1: lectura de datos

- Módulo a utilizar: acción



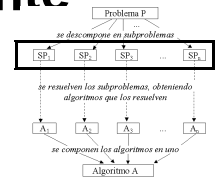
Método Descendente Ejemplo (cont.)

• Diseño – primera capa

Definimos para cada subproblema:

SP2: cálculo del área

- Módulo a utilizar: acción*



* También podría ser una función. Se deja como ejercicio

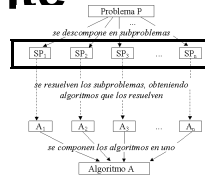


Método Descendente Ejemplo (cont.)

• Diseño – primera capa

Definimos para cada subproblema:

- SP3: mostrar área
- Módulo a utilizar: acción

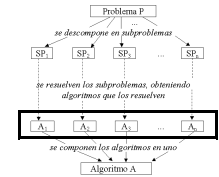


Método Descendente Ejemplo (cont.)

• Diseño – segunda capa Refinamiento

Se resuelven los subproblemas obteniendo los subalgoritmos:

- Nombre del módulo: LeerDatos
- Parámetros: $b \in \mathbb{R}$, $h \in \mathbb{R}$ de tipo resultado
- Estructuras iterativas: una estructura para validar que b sea positiva y otra para h . Una iteración como mínimo de cada una para tomar el valor de cada variable.



```

Acción LeerDatos(resultado  $b \in \mathbb{R}$ ,  $h \in \mathbb{R}$ )
Inicio
  Repetir
    Leer( $b$ )
  Hasta que  $b > 0$ 
  Repetir
    Leer( $h$ )
  Hasta que  $h > 0$ 
Fin
    
```



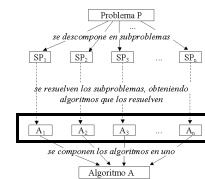
Método Descendente Ejemplo (cont.)

• Diseño – segunda capa

Refinamiento

Se resuelven los subproblemas obteniendo los subalgoritmos:

- Nombre del módulo: CalcArea
- Parámetros: $b \in \mathbb{R}$, $h \in \mathbb{R}$ de tipo dato, $a \in \mathbb{R}$ de tipo resultado
- Estructuras iterativas: no son necesarias.



```

Acción CalcArea(dato  $b \in \mathbb{R}$ ,  $h \in \mathbb{R}$ , resultado  $a \in \mathbb{R}$ )
Inicio
   $a \leftarrow (b * h) / 2$ 
Fin
    
```



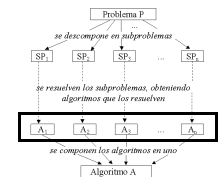
Método Descendente Ejemplo (cont.)

• Diseño – segunda capa

Refinamiento

Se resuelven los subproblemas obteniendo los subalgoritmos:

- Nombre del módulo: MostrarArea
- Parámetros: $a \in \mathbb{R}$ de tipo dato
- Estructuras iterativas: no son necesarias.



```

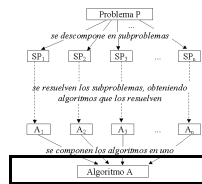
Acción MostrarArea(dato  $a \in \mathbb{R}$ )
Inicio
  Escribir( $a$ )
Fin
    
```



Método Descendente Ejemplo (cont.)

• Diseño – tercera capa Composición

Se componen los subalgoritmos en un algoritmo solo.



Definimos para el problema en general:

- Léxico: tipos que utilizaremos, nombre de cada variable con su tipo.
- Estructuras iterativas necesarias y sus características: cantidad de iteraciones (determinada o no), cantidad mínima de iteraciones (0, 1), etc.

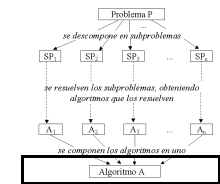


2017 Lic. Ariel Ferreira Szpiniak 57

Método Descendente Ejemplo (cont.)

• Diseño – tercera capa Composición

Se componen los subalgoritmos en un algoritmo solo.



Definimos para el problema en general:

- Léxico:

base, altura ∈ R	{base y altura del triángulo}
area ∈ R	{área del triángulo}
op ∈ [1..2]	{auxiliar}

- Estructuras iterativas: una estructura para posibilitar al usuario decidir si quiere calcular el área de otro triángulo o no. Una iteración como mínimo de cada una al menos un área.



2017 Lic. Ariel Ferreira Szpiniak 58

Método Descendente Ejemplo (cont.)

• Diseño – tercera capa Composición

Se componen los subalgoritmos en un algoritmo solo.

Algoritmo CalcularAreaTriángulo

Lexico

base, altura ∈ R	{base y altura del triángulo}
area ∈ R	{área del triángulo}
op ∈ [1..2]	{auxiliar}

Acción LeerDatos(resultado b ∈ R, h ∈ R)

Inicio

Repetir

Leer(b)

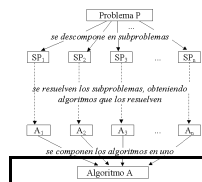
Hasta que b>0

Repetir

Leer(h)

Hasta que h>0

Fin {continúa...}



2017 Lic. Ariel Ferreira Szpiniak 59

Método Descendente Ejemplo (cont.)

• Diseño – tercera capa

Acción CalcArea(dato b ∈ R, h ∈ R, result

Inicio

a ← (b * h)/2

Fin

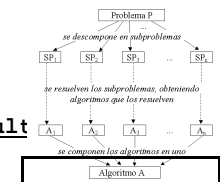
Acción MostrarArea(dato a ∈ R)

Inicio

Escribir(a)

Fin

{continúa...}



2017 Lic. Ariel Ferreira Szpiniak 60

Método Descendente Ejemplo (cont.)

• Diseño – tercera capa

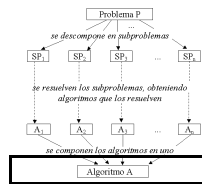
Inicio

Repetir

```
LeerDatos(base, altura)
CalcArea(base, altura, area)
MostrarArea(area)
Escribir(Presione 1 si desea continuar o 2 para salir)
Leer(op)
```

Hasta que op=2

Fin



2017 Lic. Ariel Ferreira Szpiniak 61

Método Descendente Pasos para solucionar un problema

• Análisis

El problema debe ser claramente especificado y entendido.

• Diseño

Construcción de una solución general del problema.

• Implementación

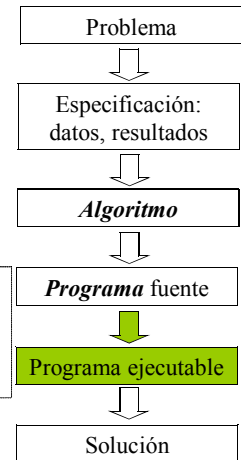
Traducción del algoritmo a un lenguaje de programación de alto nivel.

• **Compilación**

Escritura del programa Pascal, usando en nuestro caso el Compilador Turbo Pascal 7.0.

• Ejecución y Prueba

Corrida y prueba de funcionamiento del programa en la computadora.



2017 Lic. Ariel Ferreira Szpiniak 62

Método Descendente Pasos para solucionar un problema

• Análisis

El problema debe ser claramente especificado y entendido.

• Diseño

Construcción de una solución general del problema.

• Implementación

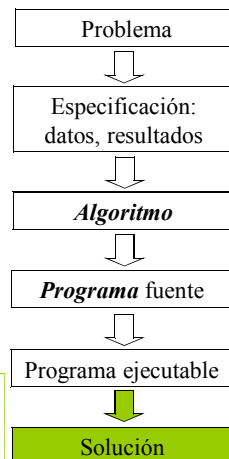
Traducción del algoritmo a un lenguaje de programación de alto nivel.

• Compilación

Escritura del programa Pascal, usando en nuestro caso el Compilador FreePascal, DevPascal o Turbo Pascal 7.0.

• **Ejecución y Prueba**

Corrida y prueba de funcionamiento del programa en la computadora.



2017 Lic. Ariel Ferreira Szpiniak 63

Bibliografía

- Biondi, J. y Clavel, G. "Introducción a la Programación. Tomo 1: Algorítmica y Lenguajes": (pags. 53-63, 181 – 190, 204-205)
- Scholl, P. y Peyrin, J.-P. "Esquemas Algorítmicos Fundamentales: Secuencias e iteración". (pags. 71 - 87)
- Watt, David: Programming Language Concepts and Paradigms, Prentice-Hall International Series in Computer Science (1990). Cap 5.
- Quetglás, Toledo, Cerverón. "Fundamentos de Informática y Programación". Capítulo 3. <http://robotica.uv.es/Libro/Indice.html>
- Programación Modular (pags 110 - 125)
- Marti Oliet, N., Segura Díaz, C., Verdejo López, J. Especificación, derivación y análisis de algoritmos. Capítulo 1 (pags 1-19). 2006
- Pascal
 - intro turbopascal.pdf (aula virtual)
 - laprogramacion en lenguaje pascal.pdf (aula virtual)
 - pascaly turbopascal.pdf (aula virtual)
 - Biondi, J. y Clavel, G. "Introducción a la Programación. Tomo 1: Algorítmica y Lenguajes": (pags. 243 - 252)
 - Joyanes Aguilar, L., "Programación en Turbo Pascal". Mc Graw Hill, 1993.
 - Wirth, N. and K. Jensen, "Pascal: User Manual and Report", 4ª ed., New York, Springer-Verlag, 1991 (traducción de la primera edición "Pascal: Manual del Usuario e Informe", Buenos Aires, El Ateneo, 1984).



2017 Lic. Ariel Ferreira Szpiniak 64

Citar/Atribuir: Ferreira, Szpiniak, A. (2017). Teoría 11: Diseño: notaciones, método descendente. Programación estructurada. Introducción a la Algorítmica y Programación (3300). Departamento de Computación. Facultad de Cs. Exactas, Fco-Qcas y Naturales. Universidad Nacional de Río Cuarto.

Usted es libre para:

Compartir: copiar y redistribuir el material en cualquier medio o formato.

Adaptar: remezclar, transformar y crear a partir del material.

El licenciante no puede revocar estas libertades en tanto usted siga los términos de la licencia.

Bajo los siguientes términos:



Atribución: Usted debe darle crédito a esta obra de manera adecuada, proporcionando un enlace a la licencia, e indicando si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo del licenciante.



Compartir Igual: Si usted mezcla, transforma o crea nuevo material a partir de esta obra, usted podrá distribuir su contribución siempre que utilice la misma licencia que la obra original.

<https://creativecommons.org/licenses/by-sa/2.5/ar/>



Programación no estructurada

Ejemplo Lenguaje de programación BASIC

2017 Lic. Ariel Ferreira Szpiniak 66

```
10 INPUT "Cuál es su nombre: "; NN$
20 PRINT "Bienvenido al 'asterisquero' "; NN$
25 PRINT
30 INPUT "con cuántos asteriscos inicia [Cero sale]: "; N
40 IF N<=0 THEN GOTO 200
50 ASS=""
60 FOR I=1 TO N
70 ASS=ASS+"*"
80 NEXT I
90 PRINT "AQUI ESTAN: "; ASS
100 INPUT "Desea más asteriscos: "; SN$
110 IF SN$="" THEN GOTO 100
120 IF SN$<>"S" AND SN$<>"s" THEN GOTO 200
130 INPUT "CUANTAS VECES DESEA REPETIRLOS [Cero sale]: ";
VECES
140 IF VECES<=0 THEN GOTO 200
150 FOR I=1 TO VECES
160 PRINT ASS;
170 NEXT I
180 PRINT
185 REM A repetir todo el ciclo (comentario)
190 GOTO 25
200 END
```

Plato de spaghetti



2017 Lic. Ariel Ferreira Szpiniak 67

```
100 REM LARGO(3,3), LARGO(3,3)
110 'Carga de datos
120 FOR I=1 TO 3
130 FOR J=1 TO 3
140 PRINT I$;J$;": ";
150 INPUT CASS(I$,J$)
160 NEXT J$
170 NEXT I$
180 'Impresión de datos
190 FOR I=1 TO 3
200 IF I=1 OR I=4 OR I=7 THEN PRINT STRING$(37,205)
    ELSE PRINT STRING$(37,196)
210 FOR J=1 TO 9
220 IF CASS(I$,J$)="" THEN PG=" " ELSE PG=CASS(I$,J$)
230 IF J=1 OR J=4 OR J=7 THEN DG="|" ELSE DG="|"
240 PRINT DG;PG;": ";
250 NEXT J$
260 PRINT "!"
270 NEXT I$ :PRINT STRING$(37,205) :STOP
300 'Determina los dígitos que NO pueden corresponder en cada casilla
310 NS=""
320 FOR I=1 TO 9
330 FOR J=1 TO 9
340 IF CASS(I$,J$)="" THEN 550
350 'Filas
360 FOR K=1 TO 9
370 IF CASS(I$,K$)="" THEN 390
380 NS=NS+CASS(I$,K$)
390 NEXT K$
400 'Columnas
410 FOR K=1 TO 9
420 IF CASS(K$,J$)="" THEN 440
430 NS=NS+CASS(K$,J$)
440 NEXT K$
450 'Modulos
460 GOSUB 1810 'Determina las primeras y ultimas fila y columna del modulo
470 FOR K=1 TO 4
480 FOR L=C1$ TO C4$
490 IF CASS(K$,L$)="" THEN 510
500 NS=NS+CASS(K$,L$)
510 NEXT L$
520 NEXT K$
530 CANS(I$,J$)=NS
540 NS=""
550 NEXT J$
560 NEXT I$
600 'Determina los dígitos que SI pueden corresponder a cada casilla
610 FOR I=1 TO 9
620 FOR J=1 TO 9
630 IF CANS(I$,J$)="" THEN 700
640 FOR K=1 TO 9
650 K$=RIGHT$(STR$(K$),1) : NP=0
660 IF INSTR(CANS(I$,J$),K$)=0 THEN NS=NS+K$
670 NEXT K$
680 CANS(I$,J$)=NS
690 NS=""
700 NEXT J$
710 NEXT I$
800 'GOSUB 2400 'Agrupa resultados parciales
900 'Procesa dígito único posible de una casilla
901 FOR I=1 TO 9
910 FOR J=1 TO 9
920 IF CANS(I$,J$)="" THEN 970
930 IF LEN(CANS(I$,J$))>1 THEN 970
```



2017 Lic. Ariel Ferreira Szpiniak 68

```

950 'GOTO 2400 'Impresión resultados parciales
960 CANS(I%,J%)=CANS(I%,J%)+P% 'marca digito ya procesado
970 NEXT J%
980 NEXT I%
1000 'Busca casillas con digito unico posible
1001 FOR I%=1 TO 9
1010 FOR J%=1 TO 9
1020 IF CANS(I%,J%)="" THEN 1040
1030 IF LEN(CANS(I%,J%))=1 THEN 900
1040 NEXT J%
1050 NEXT I%
1100 'Elimina digitos sobrantes en casilla con digito unico en fil,col y mod
1110 FOR M%=1 TO 9
1120 NS=RIGHT$(STR$(M%),1)
1130 FOR I%=1 TO 9 : D$="" : C%=0 'filas
1140 FOR J%=1 TO 9
1150 IF CANS(I%,J%)="" THEN 1180 'no procesa los nulos
1160 IF RIGHT$(CANS(I%,J%),1)=NS THEN 1180 'ni los ya procesados
1170 IF INSTR(CANS(I%,J%),NS)>0 THEN D$=D$+NS : C%=1
1180 NEXT J%
1190 IF D$=NS THEN CANS(I%,J%)=NS
1200 NEXT I%
1210 FOR J%=1 TO 9 : D$="" : F%=0 'columnas
1220 FOR I%=1 TO 9
1230 IF CANS(I%,J%)="" THEN 1260
1240 IF RIGHT$(CANS(I%,J%),1)=D$ THEN 1260
1250 IF INSTR(CANS(I%,J%),D$)>0 THEN D$=D$+NS : F%=1
1260 NEXT I%
1270 IF D$=NS THEN CANS(F%,J%)=NS
1280 NEXT J%
1290 FOR I%=1 TO 7 STEP 2 'modulos
1300 FOR J%=1 TO 7 STEP 2 : D$="" : PSC%=0 : CSC%=0
1310 GOSUB 1800
1320 FOR P%=1 TO P%
1330 FOR C%=1% TO C%
1340 IF CANS(P%,C%)="" THEN 1370
1350 IF RIGHT$(CANS(P%,C%),1)=P% THEN 1370
1360 IF INSTR(CANS(P%,C%),NS)>0 THEN D$=D$+NS : PSC%=P% : CSC%=C%
1370 NEXT C%
1380 NEXT P%
1390 IF D$=NS THEN CANS(PSC%,CSC%)=NS
1400 NEXT J%
1410 NEXT I%
1420 NEXT NS
1500 'busca casillas con 1 solo digito no procesadas
1510 FOR I%=1 TO 9
1520 FOR J%=1 TO 9
1530 IF CANS(I%,J%)="" THEN 1560
1540 IF RIGHT$(CANS(I%,J%),1)=P% THEN 1560
1550 IF LEN(CANS(I%,J%))=1 THEN 900
1560 NEXT J%
1570 NEXT I%
1600 'Imprime resultados finales
1610 FOR I%=1 TO 9
1620 FOR J%=1 TO 9
1630 IF CANS(I%,J%)="" THEN 1680
1640 IF RIGHT$(CANS(I%,J%),1)=P% THEN 1655 ELSE 1670
1650 L$=LEN(CANS(I%,J%))-1
1660 CANS(I%,J%)=LEFT$(CANS(I%,J%),L$)
1670 CANS(I%,J%)=CANS(I%,J%)
1680 PRINT USING " \ \ ";CANS(I%,J%);
1690 NEXT J% : PRINT " \ "
1700 NEXT I%

```



```

1710 GOTO 3000 'filas
1800 'fila y columna inicial y final del subcuadrado
1810 IF I%<3 THEN FI%=1 : FF%=3 : GOTO 1830
1820 IF I%<6 THEN FI%=4 : FF%=6
ELSE FI%=7 : FF%=9
1830 IF J%<3 THEN CI%=1 : CF%=3 : GOTO 1850
1840 IF J%<6 THEN CI%=4 : CF%=6
ELSE CI%=7 : CF%=9
1850 RETURN
2000 'Elimina en fil,col y mod el digito que es unico posible
2010 DGS=CANS(I%,J%)
2020 FOR K%=1 TO 9 'fila
2030 IF K%=J% THEN 2130
2040 IF CANS(I%,K%)="" THEN 2130
2050 NS=""
2060 L$=LEN(CANS(I%,K%))
2070 FOR M%=1 TO L$
2080 D$=MID$(CANS(I%,K%),M,1)
2090 IF D$=DGS THEN 2100 ELSE NS=NS+D$
2100 NEXT M%
2110 CANS(I%,K%)=NS
2120 NS=""
2130 NEXT K%
2140 FOR K%=1 TO 9 'columna
2150 IF K%=I% THEN 2240 ELSE NS=""
2160 IF CANS(K%,J%)="" THEN 2240
2170 L$=LEN(CANS(K%,J%))
2180 FOR M%=1 TO L$
2190 D$=MID$(CANS(K%,J%),M,1)
2200 IF D$=DGS THEN 2210 ELSE NS=NS+D$
2210 NEXT M%
2220 CANS(K%,J%)=NS
2230 NS=""
2240 NEXT K%
2250 GOSUB 1810 'determina primera y ultima fil y col de modulo
2260 FOR K%=FI% TO FF% 'modulo
2270 FOR R%=CI% TO CF%
2280 IF K%=I% AND R%=J% THEN 2370
2290 IF CANS(K%,R%)="" THEN 2370
2300 L$=LEN(CANS(K%,R%))
2310 FOR M%=1 TO L$
2320 D$=MID$(CANS(K%,R%),M,1)
2330 IF D$=DGS THEN 2340 ELSE NS=NS+D$
2340 NEXT M%
2350 CANS(K%,R%)=NS
2360 NS=""
2370 NEXT R%
2380 NEXT K%
2390 RETURN
2400 'Imprime resultados a medida que avanza el proceso
2410 PRINT
2420 FOR IP%=1 TO 9
2430 FOR JP%=1 TO 9
2440 PRINT USING " \ \ \ \ ";CANS(IP%,JP%);
2450 NEXT JP% : PRINT " \ "
2460 NEXT IP% : STEP
2470 RETURN
3000 END

```

