Lenguaje de Diseño en PSeInt
Introducción

RESOLUCIÓN DE PROBLEMAS
FUNDAMENTOS DE LA PROGRAMACIÓN

Ingeniería en Computación
Ingeniería en Informática
Ingeniería en Minas
Profesorado en Computación

UNIVERSIDAD NACIONAL DE SAN LUIS
DEPARTAMENTO DE INFORMÁTICA

Índice

1. Introducción

Un ***programa*** es un modelo de resolución de un problema escrito en un lenguaje de programación. De la definición anterior se desprende que escribir un programa implica:

1. Obtener una solución de un problema.
2. Expresar esta solución en un lenguaje de programación.

En general se puede decir que existe una distancia o ***diferencia*** entre lo que se podría denominar el ***lenguaje del problema*** y el ***lenguaje de programación***, en el sentido que el primero resulta menos rígido y con más posibilidades de expresión que el segundo.

El objetivo fundamental de un lenguaje de diseño es ser ***comprensible para las personas*** que van a interpretar los algoritmos escritos en él, mientras que el fin último de un lenguaje de programación es ser ***comprensible por la computadora*** que va a ejecutar el programa.

La finalidad de un lenguaje de diseño es brindar una herramienta que sirva de apoyo para el desarrollo de algoritmos. La idea es no sumar, a la complejidad del problema, las limitaciones impuestas por una notación estricta. Además, en muchas aplicaciones, es importante conseguir un algoritmo independiente del lenguaje de programación o lenguaje de implementación.

En general, cada programador, de acuerdo con su experiencia y habilidad, encontrará más expresiva una notación u otra. Imponer una notación específica, si bien, en parte implica contradecir los objetivos iniciales que justificaron el uso de los lenguajes de diseño, con la finalidad de comunicarnos, durante lo que resta del desarrollo del curso, necesitamos establecer algunas pautas para el lenguaje de diseño de algoritmos que usaremos.

Los ***objetivos básicos*** de un lenguaje de diseño son:

1. Servir de apoyo durante el proceso de resolución de un problema.
2. Servir como etapa previa al proceso de codificación. La tarea de **traducción** del lenguaje de diseño a cualquier lenguaje de programación no debería ser muy complicada.
3. En los proyectos de desarrollo de software, en los que intervienen varias personas, el lenguaje de diseño debería permitir que cada una de ellas pueda tener una visión global del trabajo de los demás, difícil de conseguir analizando directamente los programas del resto del grupo.
4. Como los lenguajes de programación proveen diferentes conjuntos de primitivas y la traducción al lenguaje de programación es posterior al diseño, podemos elegir el lenguaje de programación apropiado según el conjunto de primitivas requerido.

La intención consiste en **proponer un lenguaje de diseño de algoritmos** que sirva de apoyo para la resolución de problemas y pueda ser traducido, en forma sistemática, a un programa. El lenguaje que se usará es PSeInt.

2. Lenguaje de diseño PSeInt

PSeInt, es la abreviatura de Pseudocode Interpreter, Intérprete de Pseudocódigo. Este programa fue creado por Pablo Novara como proyecto final para el dictado de la materia Programación 1 de la carrera Ingeniería en Informática de la Facultad de Ingeniería y Ciencias Hídricas de la Universidad Nacional del Litoral. El programa utiliza pseudocódigo, una descripción de un algoritmo computacional, cuya principal misión es que el programador pueda centrarse en los aspectos lógicos de la programación, dejando el apartado técnico para cuando se vea la sintaxis de un lenguaje de programación real. PSeInt incluye en su editor diversas herramientas que permiten crear y almacenar programas en este lenguaje, ejecutarlos directamente desde su interfaz, o incluso corregir posibles defectos que se encuentre en su desarrollo.

¿Por qué usar PSeInt?

- Porque es software libre, sin necesidad de gastar dinero o violando los derechos de autor.
- Está constantemente actualizado por su creador, a diferencia de los otros compiladores e intérpretes de pseudocódigo que están descontinuados.
- Posee un foro para reportar errores y obtener ayuda que también está constantemente atendido por su creador, lo que ayuda a mejorar el programa.
- Posee una extensa ayuda, que favorece el aprendizaje del lenguaje y el utilizarlo.
- Está disponible su código fuente, y con instrucciones para ejecutarlo, de modo que se podrá personalizarlo y corregirlo.
- Posee previsualización y exportación a C, C++ y otros lenguajes, lo que ayuda a aprender estos y otros lenguajes.
- Se trata de un compilador que compila automáticamente cuando el usuario pulsa ejecutar, el algoritmo se guarda automáticamente en un archivo del disco duro, dentro de la carpeta del ejecutable PSeInt, para su posterior ejecución, haciendo más cómodo su uso.

3. Representación de Algoritmos

Previo a la definición de nuestro lenguaje de diseño, necesitaremos precisar algunos conceptos. De ahora en adelante, el procesador, como lo hemos definido en nuestro contexto, es equivalente a una computadora. La construcción del algoritmo es la etapa más dificultosa y, en éste y los próximos capítulos, daremos las herramientas básicas necesarias.

Las computadoras, como ya fue indicado, no pueden ejecutar directamente los algoritmos en forma literal como los venimos tratando. Es necesario codificarlos en un lenguaje de programación. En la mayoría de los casos, la codificación no presenta grandes dificultades ya que, los lenguajes de programación tienden, cada vez más, a la formalización que se propone, cambiando esencialmente su sintaxis.

Lo primero a considerar en el proceso de resolución de problemas es la formalización de su ***ambiente***. Vamos a definir un conjunto de reglas que nos permitirán describir, con precisión y sin ambigüedad, los ***objetos del universo de un problema***.

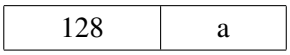
Una característica que diferencia entre sí a los objetos, es que ***cada uno*** tiene un ***nombre*** que lo identifica unívocamente, o sea, si queremos citar diferentes objetos, damos una lista de sus nombres o identificadores.

Además, cada objeto tiene un ***uso*** específico que no se puede intercambiar. Podemos decir que cada objeto tiene un ***tipo*** particular que indica características comunes a todos los estados posibles del objeto.

> Los objetos más simples con los cuales nosotros trabajaremos durante el curso son los objetos numéricos: ***enteros*** y ***reales***; los ***lógicos*** y los ***caracteres***.

Otra característica importante de los objetos es su ***valor***.

En cada instante, todo objeto del ambiente tiene un valor, para algunos objetos, este valor puede cambiar luego de la ejecución de una acción. Para resumir, podemos imaginarnos a los objetos de un ambiente como celdas rotuladas (por el nombre), donde además las celdas tienen un tamaño determinado (según el tipo) y contienen una información (un valor posible del conjunto de valores de un tipo dado)



NUMERO LETRA

3.1. Precisiones acerca de los Objetos

Los objetos se pueden clasificar en variables y constantes.

> Una ***variable*** es un objeto del ambiente cuyo valor puede cambiar y que posee además los siguientes atributos:
> un ***nombre*** que la identifica,
> un ***tipo*** que describe los valores que puede tomar la variable
> y las operaciones que con dicha variable pueden realizarse.

Cuando se define una variable, se debe precisar su nombre y su tipo. Definir una variable es crear un objeto para el procesador. En el momento de la creación de una variable, ésta tiene un valor desconocido. Para definir una variable en PSeInt se debe colocar la palabra reservada **Definir** seguido por el nombre de la variable a continuación la palabra reservada **Como** y luego el tipo.

Definir <NbreVariable> **Como** <tipo>

El <tipo> podrá ser Entero, Real, Lógico o Caracter.

> Una ***constante*** es un objeto cuyo valor no puede cambiar.

En la definición de variable hicimos referencia al concepto de ***tipo***. En realidad en la bibliografía puede encontrarse como ***tipo de datos. Dato*** es la expresión general que describe los objetos con los cuales opera un procesador. Existen diferentes tipos de datos, nosotros nos ocuparemos en este capítulo de los llamados ***tipos primitivos*** y, dentro de ellos, de los más simples: los numéricos (enteros y reales) los lógicos y los caracteres.

> El ***tipo entero***, consiste de un conjunto finito de valores enteros. La cardinalidad de este conjunto depende de las características del procesador.

> El ***tipo real***, consiste de un conjunto finito de valores reales. La cardinalidad de este conjunto también estará definida por las características del procesador.

Los números reales siempre tienen un punto decimal; las fracciones se guardan como números decimales.

> El ***tipo lógico***, también llamado ***tipo booleano***, es el conjunto de los valores de verdad: **VERDADERO y FALSO.**

El ***tipo caracter*** es el conjunto finito y ordenado de caracteres que el procesador puede reconocer.

En general, todos los procesadores reconocen el conjunto de caracteres que contiene, entre otros:

- las letras mayúsculas del abecedario .
- las letras minúsculas del abecedario.
- los dígitos decimales del $0 \ldots 9$.
- el carácter de espacio blanco, caracteres especiales tales como: $*$, $+$, $-$, $\_$, $/$, $(,)$, $,$, \$, $\wedge$, $\%$, $<$, $>$.

Una constante de tipo caracter se escribe encerrada entre COMILLAS SIMPLES, por ejemplo 'A', 'a'.

4. Expresiones y el operador de asignación

Un procesador debe ser capaz de manipular los objetos del ambiente de un algoritmo. Es decir, debe ser capaz de calcular expresiones como: $2 + 3$, $a > b$, etc. Luego:

Una ***expresión*** describe un cálculo a efectuar cuyo resultado es un valor único.

Una ***expresión*** consta de ***operadores*** y ***operandos***. Según el tipo de los objetos que manipula, se clasifican en expresiones:

- aritméticas,
- relacionales,
- lógicas.

  El resultado de una expresión aritmética es de tipo numérico, el de una expresión relacional y el de una expresión lógica es de tipo lógico.

4.1. Expresiones Aritméticas

Un operando de una expresión aritmética puede ser, por ahora, una constante de tipo numérico, una variable de tipo numérico u otra expresión aritmética, encerrada entre paréntesis. Los operadores aritméticos que soporta el **Lenguaje de diseño PSeInt** son:

Operador	Significado
+	suma
-	resta
$*$	producto
$/$	división
$\wedge$	potencia
$\%$ ó MOD	resto de la división

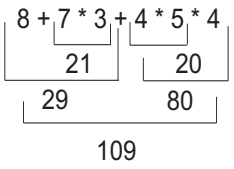
Como regla general se considera que si dos operandos tienen el mismo tipo, el resultado también es del mismo tipo. Por ejemplo, la suma de dos números enteros da como resultado otro valor entero. A continuación se dan las reglas que nos permitirán determinar cómo se evaluará una expresión de dos o más operandos:

1. Todas las operaciones que están encerradas entre paréntesis se evalúan primero, cuando existen paréntesis anidados las expresiones más internas se evalúan primero.
2. Las operaciones aritméticas, dentro de una expresión, se ejecutan con el siguiente orden o precedencia:

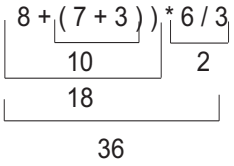
Orden de precedencia	Operadores	Significado
º	$\wedge$	potenciación (se aplica de derecha a izquierda)
2º	$*, /, \%$ ó MOD	multiplicación, división y resto (se aplican de izquierda a derecha)
3º	+, -	suma y resta (se aplican de izquierda a derecha)

La tabla anterior indica que en una expresión primero se evalúa la potenciación, luego el producto y/o la división, que tienen el mismo nivel de prioridad y, finalmente, la suma y/o resta. Tanto en el caso del producto y/o división como en el de la suma y/o resta cuando en la columna significado se dice “se aplican de izquierda a derecha” implica que si en una expresión aritmética hay seguidas tres operaciones, por ejemplo de producto, se comienza a calcular desde el que se encuentra más a la izquierda.

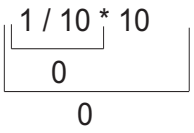
Ejemplo 1:



Ejemplo 2:



Ejemplo 3:



En el ejemplo 3 ¿pensó Ud. que el resultado correcto era 1?, tal pensamiento es incorrecto porque los operandos de esta expresión son todos enteros, luego todos los resultados, incluyendo los intermedios, deben ser enteros. Siguiendo el orden de evaluación, dado en el cuadro anterior para operadores de igual prioridad (en este caso $*$ y $/$), de izquierda a derecha antes de multiplicar se debe dividir, entonces la primera operación es $1/10$ cuyo resultado real es $0{,}1$; pero al estar trabajando con tipo entero el resultado de esta operación es 0.

Ejemplo:

Para la operación de potenciación se tiene:

$$3\ \hat{}4 \quad = \quad 81 \text{ (entero, entero} \longrightarrow \text{entero)}$$

$$3.0\ \hat{}4 \quad = \quad 81.0 \text{ (real, entero} \longrightarrow \text{real)}$$

$$2\ \hat{}3.5 \quad = \quad 11.31 \text{ (entero, real} \longrightarrow \text{real)}$$

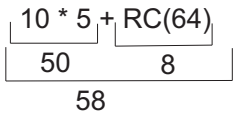
donde la notación (tipo1, tipo2 $\longrightarrow$ tipo3) se interpreta que tipo1 es el tipo del primer operando, tipo2 es el tipo del segundo operando y tipo3 es el tipo del resultado.

Además de las operaciones básicas como suma, resta, multiplicación, división y potencia, en general, existe otro conjunto de operadores especiales llamados ***funciones primitivas*** (en semejanza con las acciones primitivas) que el procesador puede ejecutar. Las ***funciones primitivas aritméticas*** del **Lenguaje de diseño PSeInt** son:

Función	Significado
ABS(X)	valor absoluto de X
RC(X)	raíz cuadrada de X
LN(X)	Logaritmo Natural de X
EXP(X)	Función Exponencial de X
SEN(X)	Seno de X
COS(X)	Coseno de X
ATAN(X)	Arcotangente de X
TRUNC(X)	Parte entera de X
REDON(X)	Entero más cercano a X
AZAR(X)	Entero aleatorio entre 0 y X-1

Al escribir el nombre de la función, a continuación, entre paréntesis, se escribe el objeto al cual se le aplicará la función, llamado ***argumento***, el procesador devolverá el resultado; por ejemplo, **RC**$(9) = 3$ y **RC**$(0{,}25) = 0{,}5$. Como las funciones primitivas devuelven valores, éstas pueden ser usadas como parte de una expresión aritmética.

Ejemplo:



4.1.1. Asignación

En un ambiente dado, para dar valor a una variable, se utiliza el ***operador de asignación***, cuyo símbolo, en el **Lenguaje de diseño PSeInt**, será ⟵, tal operación se escribe como:

$$V \longleftarrow E$$

(caracter menor + caracter menos) en la que:

1. V es el nombre de la variable a la cual el procesador va a asignarle (darle) el valor de E.
2. ⟵, identifica al operador de asignación.
3. E representa el valor a asignar y puede ser una constante, otra variable, o el resultado de la evaluación de una expresión.

Según sea el tipo de V y E, la operación de asignación aritmética se clasifica como entera o real.

En particular, diremos que $V \longleftarrow E$ es una asignación ***aritmética entera*** si:

1. V es una variable de tipo entero.
2. E es una constante entera, una variable entera, una expresión entera o una función primitiva que retorna un valor entero.

o $V \longleftarrow E$ es una ***asignación aritmética real*** si:

1. V es una variable de tipo real.
2. E es una constante real, una variable real, una expresión real o una función primitiva que retorna un valor real.

Ejemplos:

La acción $I \leftarrow$ **1** ***significa*** dar a la variable de nombre I, el valor 1.

Es ***importante remarcar*** que el número 1 ***reemplaza*** al valor que tuviere I antes de que se ejecute la acción de asignación. ***Siempre, cuando se asigna un nuevo valor a una variable, el valor anterior se pierde.***

La acción $A \leftarrow B$ ***significa*** dar a la variable A el valor de la variable B

Supongamos, por ejemplo, que el estado del ambiente **antes** de la ejecución de la acción de asignación era: A contenía el valor 6 y la variable B el valor 7. Gráficamente:

6	7
A	B

Luego de la ejecución de la acción primitiva de asignación $A \longleftarrow B$, el resultado es:

7	7
A	B

Note que mientras A pierde su viejo valor, B lo mantiene.

> La acción $A \leftarrow E$**, con** E **siendo una expresión,** ***significa*** dar a la variable A el resultado de la evaluación de la expresión E.

Ejemplo:

Sea la siguiente acción de asignación

$$\text{SUM} \longleftarrow 3{,}5 + 4{,}0 * \text{(-7.2)}$$

Primero se evalúa la expresión, con las reglas dadas para la evaluación de expresiones, luego el resultado -25.3 se asigna a la variable SUM.

> La acción $I \leftarrow I + x$ ***significa*** incrementar en x el valor de I y guardarlo en la variable I.

donde x puede ser una constante, una variable, el resultado de la evaluación de una expresión o el resultado de la evaluación de una función primitiva.

Ejemplo:

Sea la siguiente acción de asignación aritmética:

$$I \longleftarrow I + 1.0$$

Si antes de ejecutar la acción de asignación el estado del ambiente con respecto a I era que I contenía el valor 5.6, luego de la ejecución de la acción de asignación el estado del ambiente con respecto a I es que esta variable contendrá el valor 6.6. Esquemáticamente:

5.6

I

Luego:

6.6

I

Ejemplo:

$$A \longleftarrow 3 * 4 - B$$

Si el contenido de B fuere 14, luego el valor de la expresión es -2 y éste es el valor que contendrá la variable A. Es importante notar que cuando, dentro de una expresión aparece el nombre de una variable, para poder evaluar la expresión se consulta el valor de la variable, en el ejemplo la variable es B y antes de poder evaluar la expresión $3 * 4 - B$ se debe tomar el valor de B que es 14.

Por lo tanto, en una **variable** se puede, por un lado **almacenar** un valor como sucede en la variable A y, por otro, **consultar** el valor que ya tiene, como sucede en el ejemplo con la variable B.

Ejemplo:

Sean las variables A que contiene el valor 3 y B que contiene el valor -3, luego de las siguientes asignaciones: ¿Cuál será el valor de C en cada caso?

$$C \longleftarrow \textbf{ABS}(A)$$
$$C \longleftarrow \textbf{ABS}(B)$$

¿Cuál será el valor de C en cada caso?

4.2. Expresión Relacional

Para las ***comparaciones*** de valores ***numéricos*** en general o ***carácter***, los operadores relacionales con los que trabaja el **Lenguaje de diseño PSeInt** son:

Operador Relacional	Significado
=	igual
<	menor
<=	menor o igual
>	mayor
>=	mayor o igual
<>	distinto

Para la ***comparación de valores lógicos*** se usan, sólo, los operadores = y <>.

Si A es una variable entera cuyo valor es 3 y tenemos la constante entera 15, la tabla siguiente ejemplifica el cálculo de algunos posibles predicados:

Predicado	Valor
$A >= 15$	FALSO
$A = 15$	FALSO
$A < 15$	VERDADERO

Los predicados elementales o expresiones relacionales pueden combinarse, mediante los conectores u operadores lógicos para formar predicados compuestos:

Conectores u Operadores Lógicos	Significado
& o Y	conjunción
| o O	disyunción
$\sim$ o NO	negación

Para formalizar estos conceptos, sea una condición ***P*** y otra ***Q*** entonces se pueden definir las tablas de verdad para cada operador lógico como sigue:

Conjunción (y lógico)

P	Q	$P \,\&\, Q$
V	V	V
V	F	F
F	V	F
F	F	F

Como puede desprenderse de la tabla anterior, **la conjunción sólo devuelve un valor de verdad verdadero (V) cuando ambas proposiciones (P y Q) son verdaderas, en cualquier otro caso devuelve falso (F)**.

Disyunción (o lógico inclusivo)

P	Q	$P \mid Q$
V	V	V
V	F	V
F	V	V
F	F	F

La disyunción devuelve un valor de verdad Verdadero cuando al menos una de las dos proposiciones o ambas son verdaderas.

Los resultados anteriores pueden generalizarse a condiciones compuestas que contengan más de dos condiciones simples.

Negación lógica

La negación se aplica a un predicado (que puede ser simple o compuesto) y su tabla de verdad es la siguiente:

P	$\sim P$
V	F
F	V

O sea, que simplemente **cambia el valor de verdad del predicado.**

Ejemplo:

Sean X**,** K y Z tres variables enteras cuyos valores son 1, 3 y 2, respectivamente, y sean los predicados elementales P **= "**$X = 1$**",** Q **= "**$K < 2$**" y R = "**$Z = 5$**".** Veamos, entonces la evaluación de la siguiente condición o predicado compuesto (de ahora en más nos referiremos, cuando hablemos de predicados, a las condiciones).

Sea el siguiente predicado compuesto:

$$(X = 1 \mid K < 2) \& Z = 5$$

Entonces para conocer su valor de verdad se puede construir la tabla de verdad correspondiente:

p	q	r	$p \mid q$	$(p \mid q)\ \&\ r$
V	V	V	V	V
V	V	F	V	F
V	F	V	V	V
F	V	V	V	V
F	F	V	F	F
F	V	F	V	F
V	**F**	**F**	**V**	**F**
F	F	F	F	F

Para el ejemplo que estamos tratando el resultado final es **falso** tal cual se señala, remarcado en negrita, en la tabla.

Visto de otra forma más gráfica:

Es importante notar que una condición que, matemáticamente, se escribe como:

$$A < B < C$$

el procesador no la reconocerá, sino que deberá reescribirse como:

$$(A < B)\&(B < C)$$

El orden de precedencia de los operadores es:

1º $\sim$ (no lógico)
2º $\&$ (conjunción)
3º $|$ (disyunción)

Al igual que sucedía con las expresiones aritméticas, si dos operadores de igual prioridad aparecen seguidos la regla de precedencia indica que se evalúan de izquierda a derecha, por ejemplo:

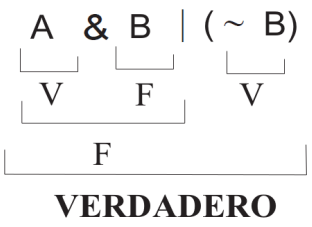
$P\&Q\&R$ se evaluará como $((P\&Q)\&R)$

4.3. Expresiones Lógicas

Un operando de una expresión lógica puede ser una variable de tipo lógica o una expresión lógica. Los operadores lógicos que soporta el **Lenguaje de diseño PSeInt** son los ya definidos en el punto anterior, es decir: $\&$, $|$, $\sim$.

Ejemplo:

Si A = **VERDADERO**, B = **FALSO** entonces la expresión $A\&B \mid (\sim B)$ da como resultado:



4.4. Operador de Asignación

Diremos que:

$$V \longleftarrow E$$

es una ***asignación lógica*** si:

1. V, es una variable lógica.
2. E, es una constante lógica **(VERDADERO, FALSO)**, una variable lógica, una expresión relacional o una expresión lógica.

Ejemplo:

Supongamos que H, T y Q son variables de tipo lógico, luego de ejecutar las siguientes acciones:

$$H \longleftarrow 2 < 5$$

$$T \longleftarrow H \mid (8 >= 9)$$

$$Q \longleftarrow \textbf{FALSO}$$

Los valores de H, T y Q son **VERDADERO, VERDADERO y FALSO** respectivamente.

$$V \longleftarrow E$$

es una ***asignación de caracter*** si:

1. V es una variable de caracter.
2. E es una constante de caracter.

5. Acciones Primitivas de Entrada y Salida de Datos

5.1. Entrada de Datos

Un valor que no pertenece al ambiente puede introducirse al mismo, mediante una acción, que llamaremos ***lectura***.

> ***Lectura***, es toda acción que permite la entrada de uno o más valores del ambiente a través de un dispositivo. Una lectura es una asignación, en el sentido que toma valores del medio externo y lo asigna a las variables del ambiente. La lectura es una acción primitiva.

En el **Lenguaje de diseño PSeInt** la lectura se denota de la siguiente manera:

LEER V

donde V es una variable del ambiente. O bien:

LEER $A, \ldots, X$

La acción **LEER** <lista de variables> asigna un nuevo valor a cada una de las variables que aparece en la lista

5.2. Salida de Datos

Un valor del ambiente puede comunicarse al mundo exterior, por ejemplo a través de la impresión sobre un papel.

> Llamaremos ***escritura*** a la acción primitiva que permite la salida de valores del ambiente a través de un dispositivo. Esta acción toma uno o más valores del ambiente y lo comunica al medio externo conservando dichas variables sus valores.

En el **Lenguaje de diseño PSeInt** la notaremos de la siguiente manera:

ESCRIBIR V

donde V es la variable cuyo valor se desea comunicar al medio externo. O bien:

ESCRIBIR $A, \ldots, X$

Esta acción comunica los valores de las variables referenciadas uno al lado del otro. La acción **ESCRIBIR** $A, \ldots, X$ se puede expresar también de la siguiente manera:

ESCRIBIR A

.....

.....

ESCRIBIR X

En este caso los valores de las variables referencias se comunican en líneas individuales (una debajo de la otra).

Es importante destacar que, en el ambiente, cuando resulte conveniente, pueden nombrarse también los Datos Auxiliares, que aunque no son los requeridos por el problema, sirven como información de la/s etapa/s intermedia/s entre los Datos de Entrada y los Resultados o Datos de Salida. De la misma manera se puede pretender comunicar información no necesariamente almacenada en una variable, por ejemplo texto, o combinaciones de variables y texto.

ESCRIBIR "El resultado es"

O también:

ESCRIBIR "El resultado es", X

6. Estructuras de Control

Estructurar el control de un conjunto de acciones, detalladas en un algoritmo, es brindar mecanismos que permitan indicar el **orden** en que las mismas van a ser llevadas a cabo.

Cuando programamos, a veces necesitamos que el programa: haga algo solo si se cumple una condición, repita una acción varias veces o elija entre varias opciones posibles. Para eso usamos las estructuras de control:

Secuencial

Condicional

Repetición

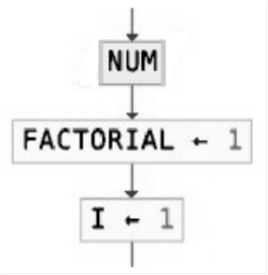
que en general no modifican el ambiente sino el orden en que el procesador ejecuta las acciones primitivas.

6.1. La Estructura de Control Secuencial

Cuando no se indique lo contrario el flujo de control de ejecución de un algoritmo seguirá la **secuencia** implícita del mismo. Entendemos por secuencia implícita, que las acciones se ejecutan en el orden en que son escritas en el algoritmo, es decir desde la primera hacia la última ("desde arriba hacia abajo"). Al terminar de ejecutarse una acción se pasa a la inmediata siguiente que está perfectamente determinada y, así siguiendo, hasta alcanzar la última acción del algoritmo. Por ejemplo, las siguientes acciones primitivas representan una secuencia:

Leer NUMERO
FACTORIAL $\leftarrow 1$
$I \leftarrow 1$

Lo mismo podría visualizarse gráficamente a través de lo que se conoce como un **diagrama de flujo** (que sirve para visualizar todos los posibles órdenes de ejecución de un algoritmo), en él las ***acciones primitivas*** que modifican el ambiente se escriben dentro de ***rectángulos***, entonces el ejemplo anterior quedaría expresado como:



6.2. La Estructura de Control Condicional

La estructura de control **condicional** permitirá que una o un conjunto de acciones primitivas se ejecuten sólo si cierta condición se cumple. Por ejemplo:

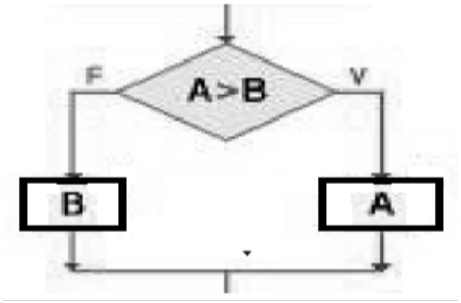
Si $n > 0$ *entonces decrementar* n *en 1*

quiere decir que sólo restaremos 1 a n si n es un número positivo. Las condiciones prevén las distintas situaciones que pueden presentarse en la resolución del problema. El condicional permite que la ejecución de cierto conjunto de acciones quede sometida a una condición.

Supongamos tener dos variables enteras, A y B, que tienen valores diferentes; buscamos escribir el mayor valor. Esto es, si $A > B$ debe escribirse el valor de A, en caso contrario el valor de B. En este caso, es evidente

que existen ***secuencias de acciones alternativas***, tales son "escribir el valor de A" o "escribir el valor de B". La elección acerca de cuál de los dos valores escribir depende de la condición: el valor de A es o no mayor que el valor de B.

El diagrama de flujo asociado con la acción condicional quedaría expresado como:



donde $A > B$ describe una condición que se debe evaluar. El resultado de esta acción completa es que el mayor de los dos valores, es el que se escribirá.

Ejemplo:

Supongamos que A tiene el valor 8 y B el valor 3. Como 8 es mayor que 3, la condición $A > B$ es Verdadera y, por lo tanto se escribirá el valor 8 como resultado de la acción primitiva **Escribir** A.

En este ejemplo sencillo hemos analizado la ***estructura de control condicional.*** El formato algorítmico de dicha estructura es:

SI <condición>
  Entonces
    <alternativa verdadera>
  SiNo
    <alternativa falsa>
FinSi

El inicio de la estructura comienza con la palabra **Si** y el final con la palabra **FinSi.**

Se llaman ***delimitadores*** a las palabras **Si, Entonces, SiNo y FinSi.**

Ahora podemos reescribir el ejemplo anterior de la siguiente manera

Si $A > B$

Entonces
Escribir A
SiNo
Escribir B
FinSi

Tanto la condición como cada una de las alternativas de la estructura, pueden ser más complejas, por ejemplo, la alternativa verdadera y la alternativa falsa pueden consistir de una secuencia de acciones primitivas en lugar de una única acción (como en el ejemplo dado).

La estructura condicional en sí misma, se considera como un ente completo (una única acción), es decir al que se ingresa en el punto en el cual la condición se evalúa. Una vez evaluada la condición se toma por el camino indicado por ***una*** de las alternativas y, luego, el control pasa a la acción primitiva que sigue al delimitador **FinSi** (si dicha acción existe).

Es necesario asegurar que las condiciones sean mutuamente excluyentes, esto es, no puede ocurrir que dos o más condiciones sean satisfechas simultáneamente, pues caeríamos en un problema de ambigüedad: ¿Cuál de los bloques de acciones es ejecutado?

Otro ejemplo:

Enunciado: Calcular la raíz cuadrada de un número, si éste no es negativo; en caso contrario no calcular nada.

Ambiente del Algoritmo:

VARIABLE	DESCRIPCIÓN
A	Variable de entrada−salida de tipo entero. Como variable de entrada contendrá el número del cual se quiere calcular la raíz cuadrada; como variable de salida, contendrá la raíz cuadrada

Algoritmo:

Versión 1:

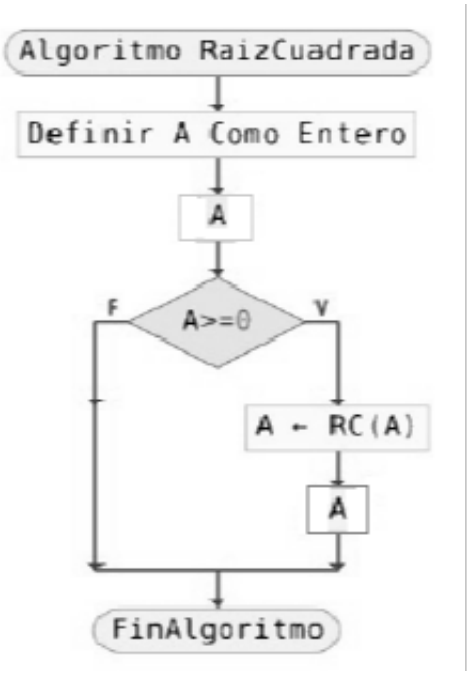
T_1 Declarar la variable A como entero.

T_2 Leer el número del cual se desea calcular la raíz cuadrada, guardándolo en la variable A.

T_3 Si el valor ingresado en T2 es positivo o cero, calcular la raíz cuadrada, usando la función primitiva correspondiente, luego informar resultado; sino no hacer nada.

Versión Final:

Algoritmo "raíz cuadrada"
Definir A **Como** Entero
Leer A //Entrada de datos
Si $A >= 0$ **Entonces** //Se determina si el valor de A es no negativo
A $\leftarrow$ **RC**(A) //Cálculo de la raíz cuadrada
Escribir A //Salida del resultado
FinSi
FinAlgoritmo



En el algoritmo anterior ***las frases u oraciones escritas entre llaves o con doble línea diagonal*** representan ***comentarios*** y sirven para documentar un algoritmo. Además el algoritmo anterior es, a la vez, un ejemplo donde un dato de entrada puede también ser utilizado como dato de salida. Puede suceder que, si la condición es falsa, no existan acciones a ejecutar (como en el ejemplo anterior). En estos casos, en la construcción algorítmica no aparece el delimitador **SINO** y el formato de la construcción condicional es:

Si <condición>

Entonces

<alternativa verdadera>

FinSi

6.2.1. Anidamiento de Estructuras Condicionales

Tanto la alternativa verdadera como la falsa, pueden contener a su vez, estructuras condicionales. Analicemos el siguiente esquema, donde: p y q, son predicados y a, b, c, y d, son acciones.

Si p

Entonces

a

SiNo

b

Si q

Entonces

c

SiNo

d

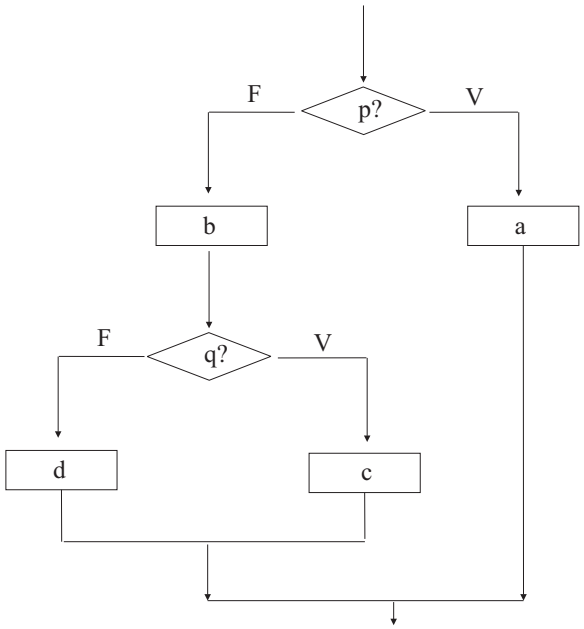
FinSi

FinSi

Veamos el resultado de todas las posibles ejecuciones de este algoritmo:

PREDICADOS		ACCIONES A EJECUTAR
p	q	
VERDADERO	**VERDADERO**	a
VERDADERO	**FALSO**	a
FALSO	**VERDADERO**	b, c
FALSO	**FALSO**	b, d

Lo cual se visualiza más fácilmente a través de un diagrama de flujo:



A continuación se ejemplifica el uso de la estructura de control condicional anidada.

Enunciado: Los operarios de una empresa trabajan en dos turnos; uno diurno, cuyo código es 1, y el otro nocturno cuyo código es 2. Se desea calcular el jornal para un operario sabiendo que, para el turno nocturno, el pago es de $5 la hora y, para el turno diurno es $3 la hora, pero en este último caso, si el día es Domingo se paga un adicional de $1 por hora.

Las fórmulas para calcular el jornal son:

fórmula 1 (para el turno nocturno) : jornal = 5 * horas trabajadas

fórmula 2a (turno diurno, no es domingo) : jornal = 3 * horas trabajadas

fórmula 2b (turno diurno, domingo) : jornal = (3 + 1) * horas trabajadas

Ambiente del Algoritmo:

VARIABLES	DESCRIPCIÓN
HORAS	Variable de entrada, de tipo entero, cuyo valor es la cantidad de horas trabajadas en un día, por un operario.
TURNO	Variable de entrada, de tipo entero, cuyo valor es el código del turno.
DIA	Variable de entrada, de tipo caracter, si su valor es 'd', indica que es día domingo, sino tiene un valor 'n'
JORNAL	Variable de salida, de tipo entero que contiene el valor de la paga que debe efectuarse.

Algoritmo:

Versión 1:

T_1 Declarar horas y turno como entero, día como caracter y jornal como entero.

T_2 Ingresar los datos necesarios horas trabajadas, tipo de turno y día trabajado.

T_3 Si el turno es nocturno calcular el jornal usando la fórmula 1, sino si es diurno pero no es domingo, usar la fórmula 2a, de lo contrario si es diurno pero es domingo, usar la fórmula 2b.

T_4 Escribir el valor del jornal calculado en T3.

Versión Final:

Algoritmo "Jornales"
Definir HORAS **Como** Entero
Definir TURNO **Como** Entero
Definir JORNAL **Como** Entero
Definir DIA **Como** Caracter
Escribir "Ingrese las horas"
Leer HORAS
Escribir "Ingrese el turno"
Leer TURNO
Escribir "Ingrese el día"
Leer DIA
Si TURNO = 2
Entonces
JORNAL $\longleftarrow 5*$ HORAS
SiNo

Si DIA <> 'd'
Entonces
JORNAL $\longleftarrow 3*$ HORAS
SiNo
JORNAL $\longleftarrow (3+1)*$ HORAS
FinSi
FinSi
Escribir JORNAL
FinAlgoritmo

La tabla siguiente describe tres ejecuciones distintas del algoritmo, a partir de distintos valores de entrada, los cuales son elegidos teniendo en cuenta que se ejecuten todas las posibles acciones del algoritmo.

Valores Iniciales	HORAS: 8 TURNO: 2 DIA: 'n'	HORAS: 12 TURNO: 1 DIA: 'n'	HORAS: 10 TURNO: 1 DIA: 'd'
Acciones Ejecutadas	JORNAL $\longleftarrow 5*8$	JORNAL $\longleftarrow 3*12$	JORNAL $\longleftarrow 4*10$
Valores Finales	JORNAL = 40	JORNAL = 36	JORNAL = 40

6.3. Estructura de Control de Repetición

En esta sección analizaremos dos estructuras de control repetitivas que permiten repetir una acción o una secuencia de acciones. Estas estructuras se diferencian, esencialmente, respecto de si el número de repeticiones es o no conocido "a priori".

6.3.1. Estructura de Repetición *"Mientras <condicion> Hacer ... FinMientras"*

El formato de esta estructura de control en **nuestro lenguaje de diseño de algoritmos** es:

Mientras <condición> **Hacer**
<secuencia de acciones>
FinMientras

En esta estructura de control la palabra **FinMientras** es un delimitador que se utiliza para indicar el fin de la secuencia de acciones a repetir. La cantidad de veces que se ejecutará la <secuencia de acciones> o el cuerpo

de la repetición: $0, 1, 2, \ldots, n$ no es conocida de antemano sino que depende de la verdad o falsedad de la <condición> . La <condición> es evaluada antes de la ejecución de la secuencia. Si el resultado es verdadero, la <secuencia de acciones> se ejecuta; si la condición resulta falsa, finaliza la repetición, es decir, se ejecutará la primer acción primitiva que siga a **FinMientras** (si ésta existe). El correspondiente diagrama de flujo es:

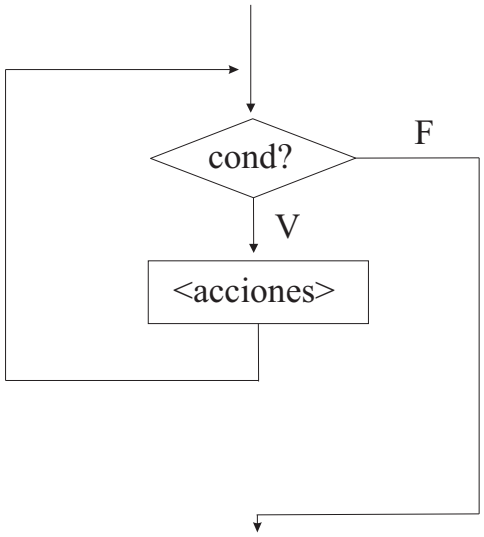


Figura 1

6.3.2. Estructura de Repetición *"Para... Hacer ...FinPara"*

Para muchas situaciones, puede resultar útil disponer de una estructura de control repetitiva que permita que una secuencia de acciones se ejecute un número fijo, conocido de antemano, de veces. El formato de esta nueva estructura de control es el siguiente:

Para $V \leftarrow Vi$ **Hasta** Vf **Con Paso** P **Hacer**
<secuencia de acciones>
FinPara

Donde:

1. V es una variable de tipo entero llamada ***variable de control*** de la repetición.
2. Vi**,** Vf **y** P pueden ser variables o constantes de tipo entero o expresiones aritméticas. Vi recibe el nombre de ***valor inicial***; Vf recibe el nombre de ***valor final y*** P ***es el paso***, o sea en cuanto se incrementa (o decrementa) V para llegar desde Vi a Vf (o desde Vf a Vi) y debe ser distinto de cero (pero puede ser positivo o negativo).

La forma en que el procesador interpreta esta estructura de control de repetición es la siguiente:

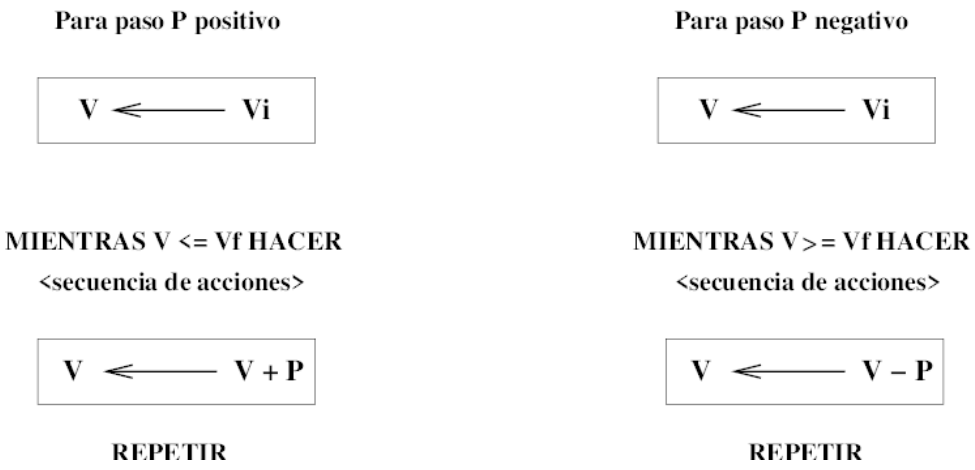


Figura 2

El número de veces que se ejecuta repetitivamente la <secuencia de acciones> se determina al momento en que el procesador ingresa a la ejecución de esta estructura. Este número está dado por la parte entera de:

$$(Vf - Vi + P)/P$$

Si este número es cero o negativo, la repetición no se ejecuta. **No es aconsejable** incluir dentro de <secuencia de acciones> acciones que modifiquen ni a V (variable de control) ni a Vi, Vf y P, aunque puedan utilizarse para efectuar cálculos.

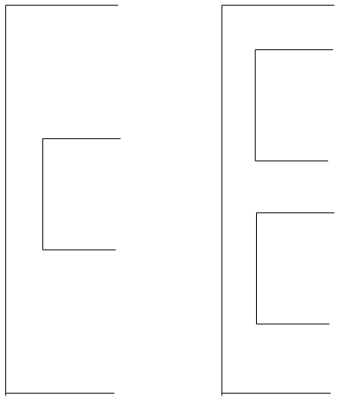
Para ejemplificar el uso de esta estructura se puede planter el problema de calcular el factorial de un número cualquiera. Por ejemplo si se quiere calcular el factorial de 4 (o 4!), es decir calcular: 1 x 2 x 3 x 4 = 24 = 4! O el factorial de 5, se calculará: 1 x 2 x 3 x 4 x 5 = 120 = 5!. El siguiente algoritmo resuelve el problema utilizando la Estructura de Repeticion "Para... Hacer ...FinPara":

Algoritmo "Factorial de n"
    Definir NUM **Como** Entero
    Definir FACTORIAL **Como** Entero
    Definir I **Como** Entero
    Leer NUM
    FACTORIAL ⟵ 1
    Para I ⟵ 1 **Hasta** NUM **Con Paso 1 Hacer**
        FACTORIAL ⟵ FACTORIAL * I
    FinPara
    Escribir FACTORIAL
FinAlgoritmo

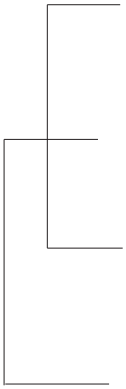
6.3.3. Anidamiento de Estructura de Repetición

De la misma forma que es posible incluir, dentro de una estructura de decisión, otra estructura condicional; también es posible insertar una estructura de repetición en el interior de otra estructura de repetición. Las reglas de anidamiento son similares en ambos casos: La estructura interna debe estar totalmente contenida dentro de la estructura externa, no permitiéndose el solapamiento.

Las figuras siguientes muestran casos válidos e inválidos de anidamiento:



Anidamientos válidos



Anidamientos inválidos

ALGORITMO "Factorial de n"
Definir N **Como** Entero
Definir J **Como** Entero
Definir NUM **Como** Entero
Definir FACTORIAL **Como** Entero
Definir I **Como** Entero
//leer cantidad de números
Leer N
Para $J \longleftarrow$ 1 **Hasta** N **Con Paso 1 Hacer**
Leer NUM
FACTORIAL $\longleftarrow$ 1
Para $I \longleftarrow$ 1 **Hasta** NUM **Con Paso 1 Hacer**
FACTORIAL $\longleftarrow$ FACTORIAL*I
FinPara
Escribir FACTORIAL
FinPara
FinAlgoritmo

En la siguiente tabla, se resume el comportamiento del algoritmo para calcular el factorial de 3 y 5.

ACCIÓN	N	J	I	NUMERO	FACTORIAL	PANTALLA
LEER N	2					
1er iteración sobre J		1^2				
LEER NUMERO				3		
FACTORIAL ⟵ 1					1	
1er iteración sobre I			1^3			
FACTORIAL ⟵ FACTORIAL * I					1	
2da iteración sobre I			2			
FACTORIAL ⟵ FACTORIAL * I					2	
3er iteración sobre I			3			
FACTORIAL ⟵ FACTORIAL * I					6	
4ta iteración sobre I			4			
ESCRIBIR FACTORIAL						6
2da iteración sobre J		2				
LEER NUMERO				5		
FACTORIAL ⟵ 1					1	
1er iteración sobre I			1^5			
FACTORIAL ⟵ FACTORIAL * I					1	
2da iteración sobre I			2			
FACTORIAL ⟵ FACTORIAL * I					2	
3er iteración sobre I			3			
FACTORIAL ⟵ FACTORIAL * I					6	
4ta iteración sobre I			4			
FACTORIAL ⟵ FACTORIAL * I					24	
5ta iteración sobre I			5			
FACTORIAL ⟵ FACTORIAL * I					120	
6ta iteración sobre I			6			
ESCRIBIR FACTORIAL						120
3er iteración sobre J		3				