PASCAL

COMANDOS IMPORTANTES:

- `cd` - Comando que permite cambiar el directorio o carpeta de trabajo, también llamado directorio corriente.
- `cd..` - Comando que permite volver al directorio o carpeta de trabajo, anterior al que nos encontramos.
- `dir` - Permite ver los archivos y subcarpetas que integran un directorio.
- `del` - Permite borrar archivos
- `mkdir` - Permite construir una carpeta nueva.
- `copy` - Permite copiar archivos
- `rename` - Permite cambiar el nombre de un archivo
- `fpc` es el nombre del compilador Free Pascal.
- `hola.pas` es el nombre del archivo que contiene el programa fuente
- `-Co -Cr` son opciones que le indican al compilador que debe generar el control de rangos, desborde de operaciones aritméticas. Es muy importante que indique estas opciones, para asegurarse que sus programas ejecutan sin estos errores.
- `-gl` esta opción hace que el compilador devuelva la línea del código fuente donde se produce el error de ejecución.
- `-Miso` esta opción hace que el compilador se comporte más parecido al Pascal usado en el libro del curso.

En términos sencillos, una computadora se puede considerar como un procesador de información. Es posible introducir datos e información a la computadora en forma de entrada(**input**) y procesarlos para producir una salida(**output**).

Entrada -> Computadora -> Salida

A los componentes físicos de una computadora se le da el nombre de **Hardware**, mientras que al conjunto de programas que se escriben para una computadora en ocasiones se le da el nombre de **Software**.
Los dispositivos de entrada/salida(E/S) permiten la comunicación entre el usuario y la computadora. Ejemplos de dispositivos de entrada son teclado, mouse. En cambio los dispositivos de salida permiten la comunicación entre la computadora y el usuario ejemplos de dispositivos de salida son las pantallas, impresoras.

Modelo refinado de una computadora:

Entrada -> Computadora (Unidad de memoria / Unidad central de proceso) -> Salida

Tipos fundamentales de instrucciones que puede ejecutar una computadora:
-Instrucciones de entrada/salida. Instrucciones para transferir información y datos entre los periféricos y la memoria principal.
-Instrucciones aritméticas y lógicas para realizar operaciones aritméticas o lógicas sobre datos almacenados.
-Instrucciones de selección que incluyen mecanismos de decisión que permiten al programa elegir diferentes cursos de acción.
-Instrucciones cíclicas que permiten repetir una secuencia de instrucciones más de una vez.
-Instrucciones de procedimientos que permiten dar nombre a un grupo de instrucciones que constituyen entonces un procedimiento.

Como lenguajes de alto nivel pueden considerarse el Ada, BASIC, COBOL, FORTRAN, Lisp, Modula-2, Pascal y PL/1. En dichos lenguajes las instrucciones de programas o proposiciones se escriben con nombres y palabras comunes que presentan los datos que se van a manipular y las acciones que se van a llevar a cabo. Se pueden escribir de tal forma que no dependen específicamente de una computadora.
Es decir que se pueden realizar en distintas computadoras a través de un **EDITOR DE TEXTO**. Los programas escritos en lenguaje de alto nivel se llaman **PROGRAMAS FUENTE**. El compilador traduce el programa fuente a un programa llamado **PROGRAMA EJECUTABLE**, que es el que se utiliza en la fase de ejecución de un programa.

PASCAL

El lenguaje de programación Pascal (en honor de **Blas Pascal**, matemático francés del siglo XVII) se debe a **Niklaus Wirth**, quien lo elaboró a finales de la década del 1960 como lenguaje de enseñanza. Por medio de Pascal es posible demostrar principios de la computación y conceptos de programación junto con la resolución de problemas en una forma muy organizada y altamente estructurada.

Programa fuente (Programa de alto nivel) **-> Compilador -> Programa objeto**(Programa de bajo nivel)

```
PROGRAM triángulo (input, output);
VAR altura, base, área : real;
BEGIN
        read (altura, base);
        área := 0.5 * altura * base;
        write (área)
END.
```

El equipo de cómputo, junto con los programas, se denomina **SISTEMA DE CÓMPUTO (sistema operativo)** . El proceso de compilar y ejecutar un programa fuente en un lenguaje de alto nivel como Pascal para obtener resultados (salida) se efectúa dentro del sistema de cómputo. Pascal obtiene datos de entrada durante la ejecución de un programa. **EL SISTEMA OPERATIVO** es un conjunto de programas que se encarga de que el usuario pueda procesar un programa en Pascal.
Para procesar un programa en Pascal primero se ordena al sistema operativo que compile el programa fuente para generar el programa objeto. En caso de que no existan errores (llamados **ERRORES DE COMPILACIÓN**) en el programa fuente, se ordena al sistema operativo que **ENLACE** (cargue) el programa objeto con cualquier **PROCEDIMIENTO DE BIBLIOTECA** (programas de uso frecuente que residen en la biblioteca del sistema) que utilice o al que haga referencia el programa en Pascal. El proceso de enlace da por resultado un **PROGRAMA EJECUTABLE**. Por último se ordena al sistema operativo ejecutar, o correr, el programa ejecutable con los datos de entrada que se proporcionan. Una vez más, si se supone que no se presentan errores durante la ejecución (llamados **ERRORES DE EJECUCIÓN**), se obtienen las salidas del programa Pascal.

Modelo sencillo de compilación y ejecución de un programa

Programa fuente en Pascal -> SISTEMA
Datos de entrada -> DE CÓMPUTO -> Salida

Compilación y ejecución

Programa fuente en Pascal -> Compilador de Pascal -> Programa Objeto -> Enlace (<- Procedimientos de biblioteca) -> Programa ejecutable -> Computadora*
Datos de entrada ->Computadora* -> Salida

La resolución de problemas se puede dividir en tres pasos principales:
1-Análisis de problemas.
2-Diseño de algoritmos.
3-Resolución en computadora.

Análisis de problemas:
1-Definición del problema.
2-Especificaciones de la entrada.
3-Especificaciones de la salida.

El primer paso, análisis del problema, requiere una definición clara del problema, el cual debe comprenderse para poder analizarlo con cuidado. A continuación, es preciso encontrar la

solución al problema. Esta solución se puede expresar en términos de un **ALGORITMO**, que es un procedimiento paso por paso para resolver el problema dado. Por último, para resolver el problema en la computadora, es necesario traducir el algoritmo a un lenguaje de cómputo (en este caso, Pascal) y después verificar o probar que el programa resuelva realmente el problema dado.

Análisis de problema

Cuando se especifica un problema que se desea resolver en la computadora, es preciso expresarlo y comprenderlo con gran claridad. El problema se debe definir perfectamente para poder llegar a una solución satisfactoria. Dado que se busca una solución en computadora, es necesario describir con detalle suficiente las especificaciones de entrada y salida del problema. Definir con precisión las especificaciones de entrada y salida son requisitos importantes para lograr una solución efectiva y eficiente.

DISEÑO DE ALGORITMOS

Los problemas complejos se pueden resolver en forma efectiva con la computadora cuando se descomponen en subproblemas que son más fáciles de resolver que el original. Esta estrategia se denomina a veces como divide y vencerás. Dicha manera de trabajar en subproblemas y que a su vez esos dividirlos en otros más pequeños se denomina **diseño descendente**. El proceso de descomponer el problema en cada etapa se llama **refinación por pasos**.

Ventajas del diseño descendente:
-El problema se comprende con facilidad y se organiza lógicamente en partes llamadas **módulos**. Así la solución del problema tiene una estructura.
-Es fácil de modificar los módulos
-Se facilita la prueba de la solución

El diseño descendente parte de una descripción general y se refina hasta que la solución tiene suficiente detalle como para traducirla a un lenguaje de computadora. Este proceso también se llama **diseño de algoritmos**, ya que realmente está diseñando la solución al problema, o algoritmo.

ALGORITMOS
Un algoritmo es un procedimiento paso por paso para resolver un problema dado.
Las instrucciones deben ser claras y sin ambigüedades, y lo bastante específicas como para ejecutarse y terminarse en un número finito de pasos.

Algoritmo del área de un triángulo
Paso 1: Introducir la altura y la base.
Paso 2: Calcular el área = ½(altura)(base).
Paso 3: Exhibir la altura, la base y el área.

Ejecución de algoritmos

La ejecución de un programa para computadora que representa a un algoritmo dado la realiza la computadora, pero en caso de existir errores en el programa o hasta en el algoritmo, es posible que el programador tenga que ejecutar el algoritmo a mano.

Cuando a un algoritmo lo escribimos como una mezcla de español e instrucciones de computadora, se dice que el algoritmo está escrito en **pseudocódigo**. Una vez que tenemos el pseudocódigo, casi siempre es una tarea sencilla traducirlo a un lenguaje de cómputo como en este caso viene siendo Pascal.

Resolución por computadora

Cuando se ha diseñado el algoritmo de un problema, se puede proceder con la fase de resolución en computadora. El primer paso es escribir, o **codificar**, el programa en un lenguaje de programación como Pascal. Luego se compila y se ejecuta el programa. Este paso incluye la corrección de errores, tanto de compilación como de ejecución. El tercer paso es verificar o probar que el programa sea correcto.

Programa de Pascal

Después de diseñar el algoritmo para un problema dado, es necesario escribir en detalle las instrucciones para la computadora. Luego de escribir las instrucciones en la computadora, es decir que se escribió el programa en Pascal y se desea continuar con la ejecución en la computadora: es preciso introducir en ella el programa. Dos estrategias comunes se conocen como **modo interactivo** y **modo por lotes**. En un **sistema interactivo**, el usuario se comunica directamente con el sistema operativo, casi siempre a través de una terminal de vídeo.
En un **sistema de procesamiento por lotes**, el usuario entrega al sistema operativo su programa y los datos de entrada que se requieran junto con todos los comandos, para que se procese el programa. El usuario debe entonces esperar por los resultados.

Un **sistema interactivo** tiene por común un programa, llamado **editor de texto**, que permite al usuario crear el programa fuente en Pascal, y cuenta con funciones de edición para modificar el programa. Es usual que el editor de texto cree un **archivo** para almacenar el programa fuente en el sistema de computo.

Compilación y ejecución de programas

Una vez creado el programa en Pascal es posible compilarlo y ejecutarlo. Durante el proceso de compilación, el compilador lee el archivo que contiene el programa fuente y lo traduce a las instrucciones reales en lenguaje de máquina que ejecuta la computadora. El conjunto de instrucciones en lenguaje de máquina se almacena normalmente en un archivo nuevo.

Durante el proceso de compilación, si el compilador detecta errores estos son similares a los errores gramaticales en español y se conocen como errores de **sintaxis**. Cuando el compilador informa al usuario que descubrió un error de este tipo, el usuario decide cual es la corrección que hay que hacer en el programa fuente, utilizando el editor de textos para corregirlo y luego volver a compilarlo.
El último paso para procesar el programa es la ejecución del programa objeto. Este paso incluye el procesamiento de todos los datos de entrada que se suministren al programa. Aun cuando el programa compile correctamente, esto no significa que el programa está libre de errores, es posible que también se presenten errores durante la fase de ejecución.
Los errores de este tipo se denominan errores de ejecución, implican la edición y compilación del programa fuente y la ejecución del nuevo programa objeto. Los errores de ejecución de los programas se conocen como errores de **semántica**. El proceso de identificar y corregir o eliminar estos errores se llama **depuración**.

Es importante hacer notar que los comandos que se utilizan para compilar y ejecutar los programas de computadora varían en los diferentes sistemas de cómputo. Por ejemplo, es posible que la computadora tenga un **intérprete** de Pascal en vez de un compilador. Cuando se utiliza un intérprete para procesar un programa en Pascal, se traduce y ejecuta cada proposición del programa fuente antes de procesar la siguiente proposición del programa fuente. Es decir, los intérpretes traducen y ejecutan las proposiciones del programa fuente una por una. Si se encuentra un error, el proceso se detiene y es posible corregir el error inmediatamente. En cambio, los compiladores traducen el programa fuente completo solo puede iniciarse la ejecución una vez eliminados todos los errores de sintaxis.

Prueba de programas

Aun así el programa no tiene errores de compilación o errores de ejecución, es posible que no esté correcto, el origen de otro posible error puede que esté en el diseño original del algoritmo. A estos errores se les llama a veces **errores de lógica.** Si el programa tiene este tipo de errores es necesario volver a la fase de algoritmo y modificarlo, cambiar el programa fuente y compilar y ejecutar una vez más. Para probar que el programa es correcto, es necesario ejecutar varias veces el programa con diversas entradas, si los resultados son correctos, aumentará la confianza que se puede depositar en el algoritmo y el programa. Sin embargo, esto no garantiza que el programa carezca de errores, ya que podría fallar con otros datos de entrada.
La prueba de los programas para determinar que sean correctos es una práctica necesaria para la creación de una exitosa solución en computadora para un problema dado.

Técnicas de prueba y depuración

Probar a mano casos pequeños y casos excepcionales es una técnica importante de prueba de algoritmos que ayuda a localizar errores de lógica y pasos faltantes en el algoritmo antes de que se hayan escrito instrucciones en el lenguaje de cómputo.

PROGRAM
La palabra program es necesaria en todos los programas de Pascal dado que es el encabezado del programa.

COMENTARIOS
Los comentarios en Pascal se escriben (*texto a comentar*) o {texto a comentar}, sirven para incluir documentación y otro tipo de información para el programador y demás personas que lean el programa fuente. El compilador pascal hace caso omiso a dichos comentarios.

VAR
La palabra Var define o declara las variables (**localidades de memoria** a las que se hace referencia por nombre) que se utilizaran en el programa. En Pascal, todos los datos a los que el programador asigna nombres se deben declarar antes de las proposiciones del programa que los utilizan.
Después de la lista de variables designadas todas mediante coma(,) de por medio, se coloca el signo(:) y se define el tipo de dato que va a manejar la variable, por ejemplo, tipo integer (enteros), tipo real (números reales) , tipo char(caracteres), tipo boolean(true/false).

PROPOSICIONES EJECUTABLES
Después de la declaración de variables están las instrucciones del programa en sí, o proposiciones ejecutables. (La declaración de variables no se considera proposición ejecutable, ya que no especifica operación alguna sobre los datos).

BEGIN Y END
Las proposiciones ejecutables del programa están encerradas entre las palabras **BEGIN y END**. Las proposiciones están separadas entre sí por signos de punto y coma(;), la última proposición antes de la palabra **end** no requiere punto y coma. **Begin y End** no son proposiciones y, por tanto, no están separadas por signos de punto y coma. Cuando se analice una sola proposición no es necesario el punto y coma que le sigue.

WRITE
Esta proposición ordena a la computadora a que exhiba el mensaje que se pone (‘Entre paréntesis y comillas simples ’).Y en caso de haber otra proposición se va a colocar pegado en la pantalla.

WriteLn
Esta proposición ordena a la computadora a que exhiba solo la línea de código y en caso de

haber más proposiciones se colocarán luego de un salto de línea en la pantalla.

ReadLn
Dadas dos variables (num1, num2) si se escribe ReadLn (num1, num2), se está indicando que los valores de las variables num1 y num2, van a ser asignados por el usuario al momento de estar en funcionamiento el programa, si el usuario coloca 3 números, el tercero va a ser desechado, además de todo el espacio vacío y el **carácter de fin de línea** (<eoln>, end of line, es un carácter ´invisible´, que marca el fin de la línea) dado que genera un salto de línea.

Read
Cumple la misma función que la proposición anterior, lo único que no genera salto de línea en la cmd, por lo cual los valores no se desechan y existe la posibilidad de que se guarden en otras variables, de proposiciones siguientes del programa.

PROPOSICIONES DE ASIGNACIÓN

Dichas proposiciones llevan 3 elementos, a la izquierda una variable, en el medio el signo (:=) y a la derecha variables ya designadas o otros valores, lo que genera esta proposición es que la variable de la izquierda tome guarde el valor de lo que se genera a la derecha.

Estructura de un programa en Pascal

Un programa en Pascal consta de dos componentes: un encabezado y un bloque de programa.
El encabezado del programa es una sola proposición que comienza con la palabra **Program**. A esta palabra le sigue el nombre que se asigna a todo el programa. Puesto que el encabezado del programa es una proposición se separa del resto del programa mediante (;).
El bloque del programa es el resto del programa y está formado por dos componentes: las declaraciones y las proposiciones ejecutables. Las **declaraciones** definen o declaran los objetos que tienen nombres (a los que se puede hacer referencia) como son las variables y las constantes. Las **proposiciones ejecutables** son las instrucciones que va a llevar a cabo la computadora cuando se ejecute el programa. Estas proposiciones constituyen la parte activa del programa, ya que ellas, y sólo ellas, pueden hacer que se realice la manipulación de los datos. El conjunto de proposiciones ejecutables está delimitado por las palabras **begin y end**. El final de todo el bloque es por medio de un punto que debe seguir a la palabra **end.**

IDENTIFICADORES EN PASCAL
Sirven para identificar los objetos que se desean manipular. Lo normal es utilizar nombres para designar cosas tales como el programa completo, las variables, algunas constantes, los procedimientos y las funciones. Los identificadores que se emplean en los programas en Pascal están formados por un conjunto de caracteres que pueden incluir letras de la A a la Z y los digitos del 0 al 9. La única restricción es que el primer carácter de cada identificador debe ser una letra. Pascal no distingue entre mayúsculas y minúsculas es lo mismo. En lo posible escribir los identificadores en no más de 8 caracteres dado que hay versiones de pascal que no

toma más de 8 caracteres para la asignación de identificadores.

PALABRAS RESERVADAS

Son palabras que están reservadas por Pascal para un uso especial como vienen siendo, Program, Var, Begin, End y Const. Dichas palabras no pueden ser elegidas como identificadores para nombres de objetos. También se encuentran los **identificadores estándar**, estos son Integer, input, output, read y write. Ya que han sido declarados previamente por el compilador. Es decir se asocian a ciertos objetos aunque no lo declare el programa.

CONSTANTES

Muchos programas contienen ciertos valores que no deben cambiar durante la ejecución del programa. Estos valores se llaman **constantes**. (Cabe recalcar que algunas variables pueden permanecer sin cambio durante la ejecución del programa, pero esto se debe únicamente a que las proposiciones ejecutables no hicieron que cambiaran. Las proposiciones ejecutables nunca pueden cambiar el valor de una constante).
La palabra **CONST** está reservada y le sigue el identificador que da nombre a la constante, un signo de igual y el valor de la constante. Las declaraciones de constantes se colocan antes de las declaraciones de variables.
Si se desea declarar más de una constante, **no** se vuelve a colocar la palabra **const** sino que se incluyen líneas adicionales de la forma identificador = constantes separadas por punto y coma:

```
Const
    interés= 0.06;
    impuesto= 0.05;
```

El valor de una constante se almacena en una localidad de la memoria de la computadora. El identificador que se emplea para dar nombre a la constante se refiere a esa localidad de memoria.

VARIABLES

Las variables son objetos de un programa que pueden cambiar de valor durante la ejecución en respuesta a las proposiciones ejecutables del programa. Se utilizan identificadores para dar nombre a las variables. Al declarar una variable se reserva una localidad en la memoria de la computadora y se etiqueta con el identificador correspondiente. Si se utiliza una variable antes de almacenar un dato en ella (o no asignarle un valor) se obtendrá un valor no definido. Mediante la asignación de un valor a la variable lo que se hace es modificar el valor de la variable, cualquiera sea el valor que estuviese guardando.
Para dar nombre a las variables en Pascal se utiliza un identificador válido. El valor de la variable puede variar, pero el nombre de la variable no. Una variable en Pascal tiene tres componentes: nombre, tipo y valor.

Para declarar una variable, es preciso especificar el nombre y el tipo de dato(entero,real,carácter,booleano) que se va a asociar a la variable. El valor de la variable queda determinado por las proposiciones que ejecuta la computadora.
Si se quieren identificar variables con distintos tipos de datos se realiza de la siguiente forma:

```
Var
    tiempo, velocidad : integer;
    distancia: real;
```

En general se puede especificar cuántas variables se desee mientras sean del mismo tipo.

TIPOS DE DATOS SIMPLES O ESTÁNDAR

Se denominan así a los tipos de datos: Integer, Real, Char y Boolean, dado que son los tipos fundamentales y de uso más común dentro de Pascal. Los tipos de datos simples describen agrupaciones de constantes o variables con diferentes características y las diversas operaciones que se pueden usar para manipular estas constantes/variables.
El número de localidades de memoria de cualquier computadora, así como el tamaño de cada localidad de memoria, es limitado, por lo que la computadora tiene un valor entero máximo que se puede representar, ya sea como constante o variable. Pascal tiene una constante entera declarada llamada **maxint**, cuyo valor es el entero más grande y **-maxint**, cuando cuyo valor es el entero más chico que se puede representar en la computadora que se usa cuando se habla del tipo de dato **integer.**
Los tipos de dato **real** si llevan punto decimal, deben tener por lo menos un dígito antes y después del punto decimal (está mal expresar .25 o 1,2).
En Pascal se pueden representar números reales con la **notación de punto flotante**, en la cual la letra e (o bien E) sustituye a “por 10 elevado a la potencia”. El número 3.21x10^-4 se escribiría como 3.21e-4 o 3.21E-4, es lo mismo (e o E).
En algunas ocasiones los tipos de datos Real son aproximaciones a valores exactos, mientras que los tipos de datos Integer son valores exactos, siempre se representan con exactitud.
Los tipos de datos **char**, en realidad se representan internamente (en la memoria de la computadora) como códigos enteros pequeños (normalmente dentro de la escala 0 a 255), pero exactamente, los caracteres son figuras comunes que se utilizan para representar nombres y direcciones.
La función **ord** se le asigna un argumento de tipo char y devuelve un integer, y **chr** es la función inversa, se le asigna un valor de tipo integer y devuelve un tipo char.
En Pascal las constantes de tipo **char** se representan por medio de un solo carácter encerrado entre apóstrofos. Ej: ‘A’ ‘a’ ‘6’ ‘{’ ‘ ’
Aquí la constante ‘A’ no es igual a la constante ‘a’. Tampoco es igual la constante char ‘6’ a la constante entera 6, para representar un espacio en blanco que va a utilizar el programa, se utiliza ‘ ’.
Las constantes del tipo **char** pueden contener una cadena de caracteres o bien un solo carácter, mientras que las variables del tipo **char** solo pueden contener un carácter, para almacenar varios se debe usar muchas variables o un arreglo.
Las constantes del tipo **boolean** se determinan como CONST ‘identificador’ = true o false

por ej: Const indicador = true;
Las declaraciones de variables que abarcan al tipo de datos booleano se construyen en forma similar a las de los demás tipos de datos simples. en la declaración.
ej: Var conmuta : Boolean;
El identificador conmuta se usa para hacer referencia a una localidad de memoria que puede almacenar solamente los valores true y false.
Operador de asignación :=, sirve para cambiar el valor almacenado en la localidad de memoria de una variable. La computadora ejecuta las proposiciones en dos pasos básicos. En el primero se determina el valor de la expresión que está a la derecha del operador de asignación. En el segundo paso almacena este valor en la localidad de memoria que corresponde a la variable cuyo nombre aparece a la izquierda del operador de asignación y se sustituye así su valor anterior.
Es posible utilizar el mismo nombre de variables tanto del lado izquierdo del operador de asignación como en la expresión del lado derecho. Es decir rapidez := rapidez + 1, es perfectamente válida, el operador de asignación no implica igualdad.
Se puede asignar un valor **integer** a una variable predefinida del tipo **real**, lo único que la computadora al encontrarse contra esto automáticamente realizará el cambio. Es decir:
Var tiempo : real;
tiempo := 60
La computadora al encontrarse con esto, automáticamente va a ejecutar tiempo := 60.0

OPERADORES ARITMÉTICOS

MOD, siempre tiene como resultado un entero positivo o cero, y el segundo operando no debe ser negativo o cero.
Para operaciones aritméticas no se permiten signos, a menos que haya sido declarada como constante con signo previamente.

FUNCIONES ESTÁNDAR

Pascal cuenta con algunas funciones estándar o incluidas. Se puede utilizar por ejemplo **sqrt** para obtener la raíz cuadrada de cualquier número no negativo. En Pascal se dice que se **invoca** (“Se llama”) una función cuando se solicita su uso.
Por ejemplo para hallar la raíz cuadrada de 100.6 lo que se hace es: raíz:= sqrt(100.6), donde la raíz es una variable previamente definida para el programa. La expresión entre paréntesis en este caso 100.6 se considera **argumento** de la función. El resultado de la función sustituye efectivamente a la invocación de la función. Es decir, la raíz:= sqrt(100.6) pasa a ser raíz:= 10.3.
La evaluación de expresiones que incluyen invocación de funciones se lleva a cabo como si se escribiera el resultado de la función en vez de su invocación. Otro ejemplo x:= 2 * sqrt(169) si queremos hallar ese valor, la proposición de asignación se maneja como si se hubiera escrito x:= 2 * 13, siendo 13 el resultado de sqrt(169).
Dos funciones más que admite Pascal son **round** y **trunc**. Estas funciones aceptan únicamente

argumentos reales, y el resultado es entero. (Se les llama muchas veces funciones de **transferencia**, ya que transfieren, o convierten, un valor de un tipo a otro).

La función **round** (redondeado) da como resultado el entero más cercano a su argumento, en caso de ser a.5 se transforma en el número que sea par. Ejemplo
round (5.5) es 6 porque 6 es par
round (4.5) es 4 porque 4 es par
round (2.1) es 2
round (-2.8) es -3

La función **trunc** da como resultado el entero que queda al omitir, o truncar, la parte fraccionaria del número real. Por ejemplo:
trunc (5.6) es 5
trunc (2.1) es 2
trunc (-2.8) es -2

La función **sqr** da como resultado un valor igual al cuadrado de su argumento con el mismo tipo que el argumento. Así que sqr(a) es tan solo una notación breve de “a” * “a, donde (a) es el argumento de la función. Ejemplos:
sqr (3) es 9 (entero)
sqr (3.0) es 9.0(real)

La función **abs** produce el valor absoluto (magnitud) de una expresión. Al igual que sqr, el tipo de su resultado es el mismo que el tipo de su argumento. Ejemplo:
abs (8) es 8 (entero)
abs (-10) es 10 (entero)
abs (5.4) es 5.4 (real)

Propiedades de las funciones estándar

-Toda función estándar ejecuta un algoritmo ya probado para calcular valores útiles.
-El tipo de argumento debe concordar con el que espera la función.
-El tipo de resultado que produce una función depende de la función de que se trate y, a veces, del tipo de argumento.
-Las expresiones que se usan como argumentos se deben evaluar (conforme las reglas usuales) antes de que se invoque la función.
-Los nombres de las funciones estándar ya están declarados (son identificadores estándar). Si se declaran explícitamente, los identificadores se referirán entonces a los objetos explícitamente declarados, es decir no se puede cambiar su valor.
-El orden en que se evalúan las expresiones determina el orden en que se realizan las invocaciones de funciones.

Puesto que las invocaciones de funciones aparecen en expresiones, es preciso obtener los valores que representan antes de que se puedan aplicar operadores a esos valores. En la expresión a * sqrt (b) / abs (c), primero se calcula el valor de sqrt (b) y de abs (c), luego de eso se realiza toda la operación.

TÉCNICAS DE PRUEBA Y DEPURACIÓN

Cuando se realizan operaciones aritméticas en Pascal, hay que tener cuidado con ciertos errores que se pueden presentar. Por ejemplo, suponga que la escala de números reales positivos permitida en un sistema de cómputo determinado es de 10^-38 a 10^38. Si un cálculo da por resultado un valor mayor a 10^38, lo más probable es que el sistema de cómputo exhiba un mensaje que indique un **desborde** (overflow) y detenga inmediatamente la ejecución del programa. En forma similar se presenta una **insuficiencia** (underflow) si el valor calculado es demasiado pequeño. La mayor parte de los sistemas no manejan la insuficiencia de la misma manera que tratan el desborde, sino que producen un resultado exactamente cero.
La precisión de la máquina se define como el número máximo de dígitos que puede emplear la computadora para representar un número. Puesto que los valores reales son números aproximados limitados por la precisión de la computadora, puede haber pérdida de exactitud, lo que podría causar errores significativos en los cálculos.

Probar y depurar cálculos aritméticos

-Asegúrese de que todas las variables se declararon.
-Asegúrese de que los tipos de datos concuerdan en las proposiciones de entrada y salida y en las proposiciones de asignación.
-Pruebense a mano los cálculos con algunos ejemplos sencillos.
-Verifiquese que la forma que se emplea para los datos de entrada y las proposiciones read y readLn sean consistentes.
-Revísese el orden en que se evaluarán las expresiones aritméticas; si se tienen dudas, utilizar paréntesis para garantizar el orden de evaluación.

Ln(x) logaritmo natural de x, siendo x entero o real y el tipo de resultado es real.
Exp(x) e^x, donde e= 2.71828… siendo x entero o real y el tipo de resultado real.

diferencia entre errores de sintaxis de semántica y de lógica

En programación, y específicamente en el lenguaje Pascal, es importante entender las diferencias entre errores de sintaxis, semántica y lógica. Aquí te explico cada uno:

1. **Errores de Sintaxis**
Estos errores ocurren cuando el código no sigue las reglas gramaticales del lenguaje de programación. Son detectados por el compilador antes de que el programa se ejecute.

Ejemplos en Pascal:
- Olvidar un punto y coma al final de una instrucción.
- Usar palabras reservadas incorrectamente.
- Incorrecta declaración de variables.

Ejemplo de código con error de sintaxis en Pascal:

```
program Ejemplo;
begin
  writeln('Hola mundo)
end.
```

Error: Falta la comilla final en el `writeln`.

2. **Errores de Semántica**

Estos errores ocurren cuando las instrucciones del programa son sintácticamente correctas, pero no tienen sentido dentro del contexto del programa. Son detectados en tiempo de compilación o ejecución.

Ejemplos en Pascal:
- Intentar asignar un valor de tipo incorrecto a una variable.
- Invocar funciones con argumentos de tipos incorrectos.

Ejemplo de código con error de semántica en Pascal:

```
program Ejemplo;
var
  x: Integer;
begin
  x := 'cadena de texto';  (* Error de semántica: asignando un string a una variable Integer *)
  writeln(x);
end.
```

Error: Asignación de un string a una variable de tipo entero.

3. **Errores de Lógica**
Estos errores son los más difíciles de detectar porque el programa compila y ejecuta, pero no produce el resultado esperado. Son errores en la lógica o en el flujo del programa.

Ejemplos en Pascal:
- Usar incorrectamente los operadores aritméticos.
- Implementar mal un algoritmo.

Ejemplo de código con error de lógica en Pascal:l

```
program Ejemplo;
var
  x, y, resultado: Integer;
begin
  x := 10;
  y := 5;
  resultado := x - y;  (* El programador quería sumar, no restar*)
  writeln('El resultado es: ', resultado);  (*Output será 5, pero se esperaba 15*)
end.
```

Error: El programador quería sumar `x` e `y`, pero usó el operador de resta en su lugar.

Resumen

-**Errores de Sintaxis**: Violación de las reglas gramaticales del lenguaje.
-**Errores de Semántica**: Uso incorrecto de elementos del lenguaje.
-**Errores de Lógica**: El programa compila y corre, pero no produce el resultado esperado debido a una falla en la lógica del programa.

Declaraciones

Las declaraciones se utilizan para definir elementos del programa, como variables, tipos, constantes, subprogramas, etc. No realizan ninguna acción cuando el programa se ejecuta, sino que sirven para establecer el contexto y los componentes que se utilizarán en el código ejecutable.

Proposiciones Ejecutables

Las proposiciones ejecutables son instrucciones que realizan acciones cuando el programa se ejecuta. Pueden incluir operaciones de asignación, llamadas a procedimientos, bucles, condicionales, etc.

Diferencias Clave

- **Propósito:**
    - **Declaraciones:** Definen y describen los elementos que el programa utilizará.
    - **Proposiciones Ejecutables:** Realizan acciones y operaciones durante la ejecución del programa.
- **Momento de Ejecución:**
    - **Declaraciones:** No se ejecutan durante la ejecución del programa; son interpretadas por el compilador para preparar el entorno de ejecución.
    - **Proposiciones Ejecutables:** Se ejecutan en tiempo de ejecución y realizan operaciones que afectan el estado del programa.

- **Ubicación:**
    - **Declaraciones:** Normalmente se colocan al principio del bloque `begin ... end` principal del programa, o dentro de procedimientos y funciones antes de las proposiciones ejecutables.
    - **Proposiciones Ejecutables:** Se colocan dentro de los bloques `begin ... end` y se ejecutan secuencialmente o según las estructuras de control del flujo del programa.

(capítulo 3)

ENTRADA, SALIDA Y RESOLUCIÓN DE PROBLEMAS

Si no hay suficientes datos en la línea de datos actual para asignar valores a todas las variables especificadas en la proposición Read y ReadLn, se pasará por alto el carácter de fin de línea y se continuará la entrada con los datos de la siguiente línea.

Si se llega al final de los datos de entrada antes de obtener un valor para todas las variables de una proposición Read o ReadLn, ocurrirá un error.

Entrada de caracteres

Read y ReadLn procesan datos de caracteres de manera similar a los datos numéricos. La diferencia es que los espacios en blanco y caracteres de tabulación y de fin de línea nunca se desechan cuando se va a leer una variable de carácter. En vez de ello, se lee el siguiente carácter de los datos de entrada y se asigna a la variable de carácter especificada en la lista de variables de la proposición Read (o ReadLn).
Por ejemplo:
Considerando que todas las siguientes variables son de tipo char.
read (car1,car2,car3,car4)
Leerá cuatro caracteres, es decir si se escribe en la cmd, 12D .
Asignaría ´1´ a car1, el carácter ´2´ a car2, ´D´ a car3 y el espacio en blanco (´ ´) a car4.

Salida

Existen dos proposiciones de salida, Write y WriteLn son proposiciones similares, la excepción es que WriteLn después de escribir todos los valores se escribe además el carácter de fin de línea.

Se puede utilizar un WriteLn sin lista de salida para hacer que se imprima o exhiba un solo carácter de fin de línea (es decir un salto de línea).

Cada uno de los tipos de datos tienen su propio campo, o grupo de columnas consecutivas (llamado **anchura de campo**), se puede especificar dicha anchura, para que no queden

espacios en blancos innecesarios.

Nótese que una secuencia larga de proposiciones write que no se vea interrumpida puede producir renglones extremadamente largos. Muchos compiladores de Pascal tienen una longitud de línea máxima que no se puede exceder.

Anchuras de campo explícitas para expresiones enteras

Pascal permite al programador especificar de manera explícita las anchuras de campo para cada expresión que se va a exhibir. Estas anchuras explícitas anulan a las anchuras de campo por omisión que se usarían normalmente. Para especificar estas anchuras de campo, cada una de las expresiones de la lista de salida va seguida de un signo de dos puntos y una expresión entera que especifica la anchura de campo que se desea utilizar para exhibir el valor de la expresión.

Ejemplo:

num1 := 4;

num2 := -12;

write (num1:5,num2:7,´Nada´ :6)

Si la anchura de campo que se especifica es demasiado pequeña para exhibir correctamente el valor de una expresión entera. Pascal ´ensancha´ automáticamente el campo para que pueda exhibir correctamente el valor, por ejemplo si se especifica una anchura de campo de 2, pero se asigna el valor -12, Pascal va a ensanchar el campo a 3, para poder mostrar el número asignado.

Anchuras de campo explícitas para expresiones de caracteres y cadenas

Pascal nunca modifica la anchura de campo especificada explícitamente para una cadena. Si la anchura es excesiva, se agregan espacios en blanco a la derecha para llenar el campo. Si los campos son demasiados pequeños para exhibir todos los caracteres de la cadena, se exhibirán únicamente los caracteres de la izquierda de la cadena hasta llenar el campo, los de la derecha se truncaran.

Ej: Write(´Mensaje´:4) Mens

Anchuras de campo explícitas para expresiones reales

En las expresiones reales se puede incluir de dos formas la anchura de campo.

-La expresión real puede ir seguida de un signo de dos puntos y una anchura, el valor de la expresión real se escribe en la forma exponencial (el valor mínimo de anchura es 6).

write (12345.67:8) _ 1.2e+04

write (12345.67:9) _ 1.23e+04

-La expresión real puede ir seguida de un signo de dos puntos y una anchura de campo y otro signo de dos puntos y una expresión entera que especifique el número total de dígitos fraccionarios que deben aparecer después del punto decimal.

Write (numreal:10:5)

WriteLn (12345.67 : 9 : 2); _12345.67

WriteLn (12345.67 : 10 : 3); _12345.670

WriteLn (12345.67 : 11 : 2); _ _ _12345.67

ReadLn, Read, Write y WriteLn, son procedimientos, pero se pueden considerar como proposiciones

Entrada y salida de archivos de texto

En un encabezado de programa como este:
Program Triángulo (input, output);
Los identificadores input y output se llaman **variables de archivo** y se usan cuando se desea transferir información entre la memoria y archivos externos. Input y Output se consideran archivos estándar en Pascal, dichos archivos se declaran automáticamente como de tipo texto cuando se listan en el encabezado del programa.
Todo archivo externo que se emplee en un programa debe contar con las variables de archivo correspondientes. Estas se deben listar, por nombre, en el encabezado del programa y también deben declararse como de tipo text (texto) en la parte de declaraciones del programa.
Sería de la forma:

Program proceso (entrada, salida)
Var
    entrada, salida: text;
A continuación es preciso poder leer y grabar estos archivos en forma similar a como se hizo con los archivos estándar input y output. Para leer un dato del archivo de texto externo asociado a la variable de archivo entrada, y asignarlo a una variable entera llamada número, se usará la proposición

read (entrada, número)

Esta proposición ordena a la computadora que lea un valor entero del archivo de texto externo asociado a la variable de archivo entrada y que almacene el valor resultante en la variable número. Puesto que la entrada se declaró como de texto, se va a leer una cadena de caracteres que represente el valor de un entero.

Un paso importante para la entrada y salida de archivos es la apertura del archivo. Si se va a leer un archivo externo, es preciso abrirlo, o conectarlo a la variable de archivo asociada, antes de poder comenzar a leer. En Pascal, la proposición **reset** realiza esta operación de apertura.
reset (entrada)
Cuando se ejecuta la proposición reset, el sistema de cómputo localizara el archivo externo apropiado (si no se puede localizar, habrá un error de ejecución) y se organizará de manera que todas las proposiciones read o readln que hagan referencia a la variable de archivo de entrada procesen el archivo externo correspondiente, a partir de su inicio.

De manera similar, la proposición **rewrite**

Rewrite (salida)

Conectará la variable de archivo salida a un archivo externo nuevo. (Si el archivo externo ya existe, se borra su contenido previo; en caso contrario, se creará un archivo externo nuevo. La imposibilidad de crear este archivo generará un error de ejecución.) Este archivo nuevo inicialmente está vacío. Es preciso usar rewrite antes de ejecutar cualquier proposición write o writeln que haga referencia a salida. Después del rewrite se puede ejecutar la proposición

write (salida,número)

para grabar en salida exactamente los mismo caracteres que se había exhibido en la terminal si se hubiera ejecutado la proposición

write (numero)

El archivo externo exacto que se asocia a una variable de archivo cuando se ejecuta un reset o rewrite varía según el sistema de cómputo que se utilice. En muchos sistemas, el **nombre de**

archivo del archivo externo es el mismo que el nombre de la variable archivo, pero pueden existir mecanismos que permitan conectar una variable de archivo con un archivo externo arbitrario.
Ejemplo:

```
Program Copiar (entdatos, saldatos);
(* Programa que copia tres números reales de la variable de archivo *)
(* entdatos a la variable de archivo saldatos *)
Var
        num1,num2,num3 : real;      (* datos reales *)
        entdatos, saldatos : text;      (* variables de archivo *)
Begin
        reset (entdatos);               (* ´abrir´ entdatos *)
        rewrite (saldatos);             (* ´abrir´ saldatos *)
        read (entdatos, num1, num2, num3);          (* obtener tres números *)
        write (saldatos, num1, num2, num3);         (* grabar tres números *)
End.
```

Puntos importantes sobre **entrada y salida de archivos de texto**

-Es posible especificar más de un archivo de entrada o salida en el encabezado del programa.
-Input y Output se declaran automáticamente como variables de archivo de tipo texto si aparecen en el encabezado del programa y se abren automáticamente para la lectura y grabación, respectivamente.
-Todas las demás variables de archivo que aparezcan en el encabezado del programa se deben declarar como de tipo texto en la parte de declaraciones de variables del programa.
-La lectura o grabación de archivos de texto externos se hace mediante proposiciones read, readln, write y writeln que especifican una variable de archivo como primer nombre de la lista de parámetros.
-Antes de ejecutar la primera proposición read o readln que utilice una variable de archivo diferente de input, es necesario abrir esa variable de archivo mediante la proposición reset.
-Antes de ejecutar la primera proposición write o writeln que utilice una variable de archivo diferente de output, es necesario abrir esa variable de archivo mediante la proposición rewrite.

En Pascal, la **impresión de eco** se refiere a la característica de un programa que muestra lo que el usuario ha escrito en la pantalla. Esto es común en programas que requieren entrada del usuario, donde el texto ingresado se muestra en la pantalla para que el usuario pueda verificar lo que está escribiendo.

Capítulo 5

SELECCIÓN

Los lenguajes de alto nivel estructurados, como Pascal, incluyen proposiciones que pueden alterar el flujo de una secuencia de instrucciones. Técnicamente, estas proposiciones se conocen como **estructuras de control**. Mediante la estructura de control **selección**, se pueden ejecutar las proposiciones de manera condicional. Es decir, si se cumple una determinada condición, se ejecuta una secuencia de proposiciones; pero si la condición no se cumple, entonces se ejecutará una secuencia de proposiciones diferente.
Proposición **IF-THEN-ELSE** corresponde a la siguiente estructura

```
IF num MOD 2 = 0
THEN writeln (´El numero es par.´)
ELSE writeln (´El numero es impar.´)
```

Expresiones booleanas

En el ejemplo anterior se probó la condición ¨num MOD 2 = 0¨ para determinar la proposición que debía ejecutar después. Dicha condición es una expresión que puede ser verdadera o falsa, según si num contiene número par o impar. Las expresiones que pueden ser o bien falsas o bien verdaderas se llaman **expresiones booleanas**.

Variables booleanas

Una expresión booleana en Pascal puede ser una variable booleana. A dichas variables sólo se les puede asignar el valor de true o false.

```
Var conmuta : Boolean;

conmuta := true        (* asigna el valor true a la variable*)

conmuta := false       (* asigna el valor false a la variable*)
```

En Pascal estándar los valores de una variable booleana no se pueden obtener de los datos de entrada mediante la proposición read.

Operadores Relacionales

=	Igual a
<>	Diferente de
<=	Menor o igual que
>=	Mayor o igual que
>	Mayor que
<	Menor que

Se pueden evaluar variables de tipo char, ´´calif > ´A´´´ es una expresión booleana válida. En este caso el operador relacional se refiere al ordenamiento de los caracteres en el sistema de cómputo, conocido como **ordenamiento lexicográfico** o **secuencia de ordenamiento**. Si el carácter almacenado en la variable calif aparece después del carácter ´A´ en el ordenamiento del sistema de cómputo de que se trate, la expresión booleana se cumplirá, en caso contrario será falsa.

Requisitos de ordenamiento de los caracteres:

-Los valores de carácter que representan a los dígitos del cero al nueve deben quedar en el orden esperado sin caracteres intermedios. Es decir ´0´ < ´1´, ´1´ < ´2´, … , ´8´ < ´9´. Aunque las relaciones son las mismas que en el caso de los enteros y los reales, estos son caracteres no numéricos.
-Las letras mayúsculas de la A a la Z deben aparecer en el orden esperado (´A´ antes de ´B´, ´B´ antes de ´C´ y así sucesivamente), pero pueden existir caracteres intermedios.
-Si se dispone de caracteres de minúsculas, deben obedecer la misma regla de ordenamiento que se requiere para los caracteres alfabéticos en mayúsculas. Esta regla no requiere que los caracteres de minúsculas aparezcan antes de las letras mayúsculas. Así, no es posible dar por sentado, en general, que ´a´ < ´A´. Nótese que estas reglas implican que ´A´ > ´2´ pudiera ser verdadero o falso, dependiendo del ordenamiento lexicográfico que se utilice en el sistema de computo.
Al usar operadores relacionales, los valores que se comparen deben ser del mismo tipo de datos, con la excepción de que es posible comparar enteros con reales.
Las siguientes expresiones booleanas no son válidas:
5 > false ‘B’ = 5.0 ‘0’ >= 0
Las expresiones de tipo de datos real pueden provocar problemas cuando se usan con operadores relacionales (dada la precisión limitada de la aritmética de reales en computadoras). Los demás tipos de datos simples (entero, de carácter y booleanos) se llaman **tipos de datos ordinales**. Estos tipos de datos se llaman ordinales porque los valores están ordenados y se pueden especificar mediante una lista. Los valores booleanos true y false están ordenados y se pueden listar. Los valores booleanos true y false están ordenados ya que false < true en Pascal.

Circuito corto

Free Pascal evalúa usando circuito corto:
-Para evaluar (E1 and E2)
-Se evalúa E1. Sea b1 su valor. Si b1 es false no se evalúa E2 y el resultado es false.
-Si b1 es true se evalúa E2. Sea b2 su valor. El resultado es b2.

Operadores booleanos

Las expresiones booleanas se pueden combinar para formar expresiones más complejas mediante los tres operadores booleanos (o lógicos) **AND** ('y'), **OR** ('o') y **NOT** ('no').
'1 < num < 10' no es una expresión legal en pascal; sin embargo (1 < núm) and (núm < 10) si es correcta.
La expresión 'P OR Q', donde P y Q son expresiones booleanas, es verdadera cuando o bien P es verdadera o bien Q es verdadera.
(5 > 10) OR ('A' < 'B')
es verdadera, ya que ('A' < 'B') es verdadera.

Los operadores de suma, resta, multiplicación y división se llaman operadores **binarios**, ya que se aplican a dos operadores. De igual manera, los operadores booleanos AND y OR se consideran operadores binarios, puesto que también ellos se aplican a dos operadores.
Los operadores que se aplican a un solo operando se llaman operadores **unarios**.
Por ejemplo, si j es una variable entera, entonces la expresión -j utiliza el operador unario (-). El operador booleano NOT también es un operador unario que invierte el valor lógico de su operando. Por ejemplo, la expresión booleana 'NOT (5>10)' es verdadera, ya que '(5>10)' es falsa.

Orden de los operadores

Los operadores booleanos y los operadores relacionales tienen un orden de prioridad.
Por tanto, en las expresiones que emplean más de uno de los tres operadores booleanos, el orden de prioridad es primero **NOT**, después **AND** y por último **OR**. Como en las expresiones aritméticas, es posible usar paréntesis para forzar la evaluación de una expresión en cualquier orden deseado.
En Pascal suelen surgir problemas cuando las expresiones booleanas incluyen tanto operadores relacionales (<, >; etc) como operadores booleanos (AND, OR y NOT). Los operadores relacionales tienen más baja prioridad que los operadores booleanos.
NOT 4 > 5
es errónea, ya que se aplica primero el operador NOT al operando entero 4. El uso de paréntesis es necesario en este caso para obtener el resultado deseado.
NOT (4 > 5)

Precedencia

1) not, +, - (unarios)
2) *, /, div, mod,and (multiplicativos)
3) +, -, or (aditivos)
4) = , <>, <, <=, >, >=, IN (relacionales)

Selección mediante la proposición IF

La proposición **IF-THEN-ELSE** tiene la siguiente estructura
IF expresión booleana
THEN proposición-1
ELSE proposición-2

La proposición comienza con la palabra reservada IF seguida de una expresión booleana. Esta va seguida de la palabra reservada THEN y una proposición en Pascal (o grupo de proposiciones). Por último, se escribe la palabra reservada ELSE, seguida también de una proposición (o grupo de proposiciones) en Pascal. No se escribe punto y coma inmediatamente antes del ELSE.
La proposición IF-THEN-ELSE se ejecuta de la siguiente manera. Se evalúa la expresión booleana, lo que produce un valor true o false. Si la expresión es verdadera, se ejecutará la proposición 1 y se hará caso omiso de la proposición 2. Si la expresión es falsa, se pasará por alto la proposición 1 y se ejecutará la proposición 2. Nótese que la proposición IF-THEN-ELSE es una sola proposición de Pascal y debe estar separada de cualquier proposición que le siga mediante un signo de punto y coma.
Pascal exige que, si se coloca más de una proposición en la parte THEN o ELSE de una proposición IF-THEN-ELSE deben estar agrupadas y delimitadas por las palabras BEGIN y END. Ejemplo

```
IF num1 > num2 THEN
BEGIN
  writeln (‘el numero mas grande es ‘, num1);
  max := num1
END
ELSE
  writeln (‘el numero mas grande es ‘, num2);
```

Como siempre, las proposiciones individuales que se agrupan con las palabras BEGIN y END se separan con signos de punto y coma, pero no se utiliza punto y coma antes del ELSE. Cuando una proposición IF-THEN-ELSE le sigue otra proposición, debe escribirse un signo de punto y coma para separar de la proposición subsecuente.

Proposición IF-THEN

Sirve para que se cumpla cierta secuencia de proposiciones únicamente si se cumple cierta condición, sino se continuará con la siguiente proposición. La proposición IF-THEN es una variante de la proposición IF-THEN-ELSE. De hecho, la proposición IF-THEN es totalmente equivalente a la siguiente proposición:

```
IF expresión booleana THEN
  proposición
ELSE;
```

La 'Proposición' que sigue al ELSE es realmente una proposición de Pascal, se llama **proposición vacía** y se incluye específicamente para utilizarse en situaciones de este tipo. Aunque en la mayor parte de los casos el uso de la proposición vacía es opcional, se verá que en algunas ocasiones es indispensable.

Proposiciones IF anidadas

Las proposiciones IF se pueden colocar dentro de otras proposiciones IF (es decir, **anidarse** en ellas). Ejemplo

```
IF num1 >= num2 THEN
  IF num1 = num2 THEN
    writeln ('ambos números son iguales a ', num1)
  ELSE
    writeln ('El numero mas grande es ', num1)
ELSE
   writeln ('El numero mas grande es ', num2)
```

Este ejemplo utiliza una proposición IF anidada. Las proposiciones IF anidadas se pueden volver cada vez más complejas y difíciles de leer.
ELSE pendiente se hace notar la regla de que un Else siempre corresponde a un THEN precedente más cercano que no tenga ya un ELSE propio. Para lograr tener un IF-THEN dentro de un IF-THEN-ELSE se debe tener la proposición IF-ELSE dentro de una pareja BEGIN-END, esto aísla de forma efectiva la proposición del proceso de asociación que se emplea para ligar el ELSE con el THEN más cercano.

Selección mediante la proposición CASE

Supóngase que se está resolviendo un problema que requiere una selección entre varias alternativas. Esta selección se podría realizar usando muchas proposiciones IF anidadas, pero Pascal cuenta con un mecanismo más apropiado: la proposición **CASE**.
La proposición CASE en Pascal es una estructura de control que permite la elección de un curso de acción de entre una lista de varias opciones. La proposición IF-THEN-ELSE permite la

selección entre dos alternativas, pero CASE permite más de dos.
Supóngase que se desea convertir una calificación en su equivalente en letra, teniendo la variable califletra como de carácter y puntos como real.
Utilizando la proposición CASE el codigo quedaria de la siguiente manera:

```
CASE califletra OF
        'A': puntos := 4.0;
        'B': puntos := 3.0;
        'C': puntos := 2.0;
        'D': puntos := 1.0;
        'F': puntos := 0.0;
END
```

La palabra reservada CASE va seguida de una expresión llamada **selector**. El selector solamente puede tener un valor que sea de un tipo de datos ordinal (es decir no puede ser real). El valor del selector determina la proposición que se va a ejecutar. Después de la palabra reservada **OF** sigue una lista de proposiciones, cada una de las cuales está etiquetada con una constante del mismo tipo de datos que la expresión del selector. La proposición CASE termina con la palabra reservada END, pero no requiere un BEGIN.
Si el valor de la expresión ordinal (selector) no concuerda con una de las etiquetas de constante, el resultado no se puede predecir, en algunos sistemas de Pascal se presentará un error durante la ejecución del programa, mientras que en otros sistemas se podrían pasar por alto todas las proposiciones del cuerpo de la proposición CASE sin dar al usuario indicación alguna de que se hizo esto. Free pascal cuenta con la posibilidad de utilizar la palabra ELSE para una proposición que se ejecutará en caso de que ninguna de las demás etiquetas concuerde con el valor del selector.

Puntos importantes de la proposición CASE:

-Si dos o más constantes se hace referencia en la misma proposición, se deben separar las constantes con comas.
-Una etiqueta constante dada no puede aparecer dos veces o corresponder a dos o más proposiciones diferentes.
-El orden de las etiquetas constantes es arbitrario.
-Todos los valores que puedan resultar de la evaluación del selector se deben especificar en la lista de etiquetas constantes. Si no se requiere acción alguna para un valor determinado del selector, entonces ese valor debe aparecer como etiqueta constante de una proposición vacía.

Estructura de los Bloques `begin ... end`

En Pascal, `begin` y `end` se utilizan para agrupar múltiples sentencias en un bloque. Esto es necesario cuando un `if`, `else`, o cualquier otra estructura de control tiene más de una sentencia que debe ejecutarse como parte de ese bloque.

Usa `begin` y `end` para agrupar múltiples sentencias que se deben ejecutar como parte de una sola condición `if`.
Si solo hay una sentencia, `begin` y `end` no son necesarios, pero es una buena práctica para mantener la claridad.
Asegúrate de que cada bloque `begin` tenga su correspondiente `end` para evitar errores de sintaxis.

capítulo 6

LA PROPOSICIÓN WHILE

La proposición **while** en Pascal ejecuta el cuerpo del ciclo **mientras** (while) se cumple la condición **hacer** (**do**). Un ejemplo de estructura de control de ciclos que tiene la siguiente forma general:

```
While expresión-booleana DO
  proposición-1;
  proposición-2
```

La palabra reservada WHILE va seguida de una expresión booleana y después de la palabra reservada DO. Mientras se cumpla la expresión booleana, se ejecutará la proposición-1 (o un grupo de proposiciones delimitado por una pareja BEGIN-END). La proposición WHILE se ejecutará en forma repetida hasta que, al evaluarse la expresión booleana, resulte ser falsa, momento en el cual el control pasará a la proposición que sigue a la proposición WHILE (proposición-2).

```
Contador :=0;
read (num);
While num > 0 DO
Begin
contador := contador + 1;
read (num)
End;
WriteLn (‘Numero de enteros positivos = ’, contador:0)
```

Cuando se ejecuta esta secuencia de proposiciones en Pascal se hace lo siguiente: se ajusta el contador inicialmente a cero, se lee un entero (num), se ejecuta el ciclo WHILE en forma repetida en tanto num sea positivo. Este segmento de código en Pascal cuenta el número de enteros positivos que se capturan como datos de entrada. Es importante comprender que la proposición read dentro del cuerpo del ciclo es esencial: sin ella el programa sería un **ciclo infinito** (sin fin), es decir, el programa nunca podría salir del ciclo. Para verificar el número de veces que se ha ejecutado el cuerpo del ciclo, puede utilizarse una variable llamada **variable de control de ciclo** para lograrlo. El siguiente segmento en Pascal realiza ese trabajo. (suponga que i, contador y num se declararon como enteros):

```
Contador := 0;      (*Inicializar el contador *)
i := 1;                    (* Inicializar la variable de control de ciclo*)
While i <= 20 Do       (* Continuar hasta que i > 20. *)
Begin
  read(num);             (*Capturar el siguiente dato de entrada*)
  if num > 0 Then
    contador := contador + 1;
  i := i + 1                (*Incrementar la variable de control del ciclo*)
End;
WriteLn('El numero de enteros positivos es', contador:1)
```

Es necesario asignar un valor inicial a la variable de control de ciclo i (en este caso uno) porque la computadora no es capaz de determinar el valor de i <= 20 sin conocer el valor de i. Es preciso cambiar el valor de la variable de control de ciclo dentro del cuerpo del ciclo para evitar un ciclo infinito. En este caso, cuando i = 21 el ciclo no se ejecutará más.

CENTINELAS Y CICLOS

Un **centinela** es un valor especial que se emplea para indicar el final de una lista de datos. Por ejemplo, suponga que se tiene una lista de calificaciones de estudiantes (cada una en la escala de cero a 100). El número de calificaciones no se conoce, pero la lista termina con el número -999(el centinela).

```
total := 0;
read (calif);
While calif <> -999 Do
Begin
  total := total + calif;
  read (calif)
End;
WriteLn ('El total de las calificaciones es ',total:0)
```

Se puede usar una proposición WHILE para controlar el cuerpo del ciclo que obtiene las calificaciones de los datos de entrada. La expresión boolean de la proposición WHILE se utilizará para probar si ya terminó la lista mediante la detección del centinela.

CONTROL DE CICLOS MEDIANTE VARIABLES BOOLEANAS

Es posible incluir expresiones booleanas compuestas en las proposiciones While de Pascal mediante los operadores booleanos AND, OR y NOT. Las variables booleanas (en ocasiones llamadas **indicadores**) se pueden usar para controlar la ejecución de una proposición While. Ejemplo:

```
read(num);
indic := false;
divisor := 2;
While Not indic And (divisor < num) Do
Begin
  indic := num Mod divisor =0;
  divisor := divisor + 1
End;
Write('El numero', num:1,' ');
If indic Then
  WriteLn('Tiene divisor exacto.')
Else
  WriteLn('No tiene divisor exacto.')
```

indic := num Mod divisor = 0; es una expresión booleana y producirá el valor true o false al evaluarse; el valor resultante se asigna entonces a la variable booleana indic.

Puntos importantes de la proposición WHILE

-Se debe asignar un valor inicial a todas las variables de control del ciclo antes de comenzar este.
-La expresión booleana de la proposición **while** se evalúa antes de entrar al ciclo.
-La variable de control del ciclo se debe modificar dentro del ciclo para evitar un ciclo infinito.

Proposiciones WHILE anidadas

Recuérdese que las proposiciones dentro del cuerpo de un ciclo WHILE pueden ser cualquier proposición o grupo de proposiciones en Pascal. De manera específica, se puede incluir otra proposición WHILE.

OTRAS ESTRUCTURAS CÍCLICAS

Es posible que se presenten situaciones en las que se desea que un ciclo se ejecute por lo menos una vez antes de probar si se debe repetir o no la ejecución del ciclo.
En la proposición WHILE, si el valor de la expresión booleana inicialmente es falso, el cuerpo del ciclo no se ejecutará. Pero existe la proposición **REPEAT-UNTIL**. En estos ciclos se ejecuta primero el cuerpo del ciclo y en seguida se evalúa la expresión booleana.

PROPOSICION REPEAT-UNTIL

La proposición REPEAT-UNTIL, **repite** (repeat) el cuerpo del ciclo **hasta que** (until) se cumple la condición, tiene la siguiente forma general:

```
REPEAT
 proposición;
 proposición;
 .
 .
 .
 proposición
UNTIL expresion booleana
```

La palabra reservada **REPEAT** va seguida de una proposición en Pascal que forma el cuerpo del ciclo. Esta proposición se ejecuta en forma repetida hasta que el valor de la expresión booleana sea true. Así, el ciclo REPEAT-UNTIL se ejecuta en tanto el valor de la expresión booleana sea false, exactamente al contrario que la proposición WHILE. Como siempre, es posible colocar un grupo de proposiciones en el cuerpo del ciclo, pero no necesita una pareja BEGIN-END después de la palabra REPEAT si se especifica más una proposición. Las palabras reservadas REPEAT y UNTIL realizan de manera efectiva la función de delimitación.
Ejemplo:

```
contador := 1;
Repeat
  read (num);
  contador := contador + 1
Until contador > 20;
WriteLn('Se leyeron veinte números');
```

En esta secuencia de proposiciones, el cuerpo del ciclo Repeat se ejecutará hasta que el contador pase de 20. Esto sucederá después de 20 ejecuciones del cuerpo del ciclo, y luego se ejecuta la proposición writeln. Obsérvese que no se requiere una pareja BEGIN-END.

Ley de DeMorgan

En general, si P y Q son expresiones booleanas, entonces “NOT (P AND Q)” es lógicamente idéntica a “NOT P OR NOT Q”. Además, “NOT (P OR Q)” es lógicamente idéntica a “NOT P AND NOT Q”.

Usar el ciclo REPEAT-UNTIL en la validación de datos. En este caso el programa espera un número de mes en la escala de 1 a 12. Si el valor que se obtiene del dispositivo de entrada es incorrecto, se exhibe un mensaje apropiado para el usuario y se captura un valor adicional.

```
write (‘Por favor escriba el número del mes: ’);
REPEAT
  ReadLn(mes);
  IF (mes < 1) OR (mes >12) THEN
    Write (‘Favor de dar un valor entre 1 y 12: ’)
UNTIL (mes >= 1) AND (mes <= 12)
```

Esta técnica cíclica de verificación es muy útil cuando se emplea un sistema de cómputo interactivo, ya que permite la corrección inmediata de un error de entrada.

LA PROPOSICIÓN FOR

Algunas veces se sabe por adelantado el número exacto de veces que se desea ejecutar las proposiciones de un ciclo. En este caso se puede usar la proposición **FOR** (**para**). La proposición FOR en Pascal es una estructura de control cíclica que ejecuta el cuerpo de un ciclo un número específico de veces y lleva automáticamente la cuenta del número de veces que se “pasa” por el cuerpo del ciclo.
Por ejemplo, la siguiente proposición FOR hará que se lean n enteros y se calcule su suma:

```
total := 0;
For i := 1 To 20 Do
Begin
  read (num);
  total := total + num
End
```

Nótese que el valor de la variable de control del ciclo FOR (también llamada muchas veces **contador**) queda sin definir cuando termina el ciclo. (No sucede así en el caso de los ciclos WHILE; el valor de la variable de control si se conoce cuando se termina el ciclo). Cualquier variable de un tipo ordinal se puede emplear como variable de control de ciclo. La forma general de la proposición FOR es:

FOR(**para**) contador := valor inicial TO(**a**) valor final DO(**hacer**) proposición

La proposición debe comenzar con la palabra reservada FOR a la cual le sigue algo que se parece a una proposición de asignación que le da el valor inicial a la variable de conteo. Esta va seguida de la palabra reservada TO y el valor final del contador. Por último aparece la palabra reservada DO, seguida de la proposición (o grupo de proposiciones delimitado por una pareja BEGIN-END) que constituye el cuerpo del ciclo. La variable de conteo, el valor inicial y el valor final deben ser todos del mismo tipo ordinal.

FOR letra := 'A' TO 'E' DO Write(letra)

Esta vez el ciclo se ejecuta cinco veces y la salida incluirá los caracteres ABCDE.
El sucesor de un valor de carácter es el carácter que le sigue inmediatamente en el ordenamiento lexicográfico (o secuencia de ordenamiento) del sistema.

En un ciclo FOR también es posible decrementar la variable de conteo. Se emplea la palabra reservada **DOWNTO**(**hasta**) en vez de TO al escribir la proposición FOR. Por ejemplo:

FOR i:= 10 DOWNTO 0 DO Write (i)

Esta exhibirá los valores 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 y 0 en ese orden.

Para que se ejecute el cuerpo del ciclo FOR-TO-DO es preciso que el valor inicial sea menor o igual que el valor final. Del mismo modo, para que se ejecute el cuerpo de un ciclo FOR-DOWN-TO es necesario que el valor inicial sea mayor o igual que el valor final. No es un error especificar un ciclo FOR cuyo cuerpo jamás se ejecuta; en muchos casos el valor inicial o el final queda especificado mediante una expresión cuyo valor no se conoce antes de ejecutar el programa. Ejemplo:

read(num);
FOR i := 10 TO num DO write (i)

Si num es menor que 10, el cuerpo del ciclo FOR no se ejecutará.
Es importante subrayar que cualquier proposición FOR se puede sustituir por una serie equivalente de proposiciones de asignación y una proposición WHILE.
Pero no toda proposición WHILE puede sustituirse por una proposición FOR equivalente, ya que las proposiciones FOR deben especificar los valores iniciales y finales de la variable de control del ciclo.

PUNTOS IMPORTANTES DEL FOR

-La variable de control de un ciclo FOR puede ser de cualquier tipo ordinal (es decir, no real) y los valores iniciales y finales deben ser del mismo tipo.
-La variable de control del ciclo se debe declarar como variable local dentro del procedimiento que contenga al ciclo. (* En Pascal, es común declarar esta variable dentro del procedimiento o

función donde se utiliza, lo que la convierte en una **variable local**.*)(*Hasta que no veamos procedimientos ignorar*)
-La variable de control del ciclo no se puede cambiar dentro del ciclo.(*Hasta que no veamos procedimientos ignorar*) Además, no puede ser un parámetro variable formal para un procedimiento que contiene al ciclo FOR ni pasarse como parámetro variable verdadero a cualquier procedimiento que se invoque dentro del cuerpo del ciclo FOR.
-Los valores iniciales y finales de la variable de control de un ciclo FOR se pueden modificar dentro del cuerpo de un ciclo FOR sin cambiar el número de veces que se ejecuta el ciclo. Esto se debe a que los valores iniciales y finales se determinan y se guardan en variables “secretas” antes de la primera ejecución del cuerpo del ciclo. Por ejemplo, el siguiente ciclo exhibe los enteros del 1 al 10 aunque se modifica el valor final dentro del cuerpo del ciclo.

```
final := 10;
FOR i:= 1 TO final DO
BEGIN
  final := 999
  write(i)
END
```

-Después de la ejecución de una proposición FOR, el valor de la variable de control queda totalmente indefinido. Es preciso asignar otro valor a la variable de control antes de intentar usarla en una expresión.
-Si el valor inicial y final son iguales, el ciclo se ejecuta exactamente una vez. Por ejemplo la siguiente proposición FOR exhibirá únicamente el número 1.

```
FOR i:= 1 TO 1 DO write (i)
```

-Los valores iniciales y finales para un ciclo FOR son expresiones arbitrarias que pueden incluir variables, constantes y operadores. Por ejemplo, el cuerpo del siguiente ciclo FOR se ejecuta exactamente seis veces.

```
bajo := 0;
alto := 3;
FOR indice := bajo + 1 TO 2 * alto DO
BEGIN
  read (num);
  write(num)
END
```

-En este caso el valor de la variable índice irá de uno a seis. Los ciclos FOR se pueden anidar, al igual que las proposiciones WHILE y REPEAT-UNTIL. Así, las siguientes proposiciones FOR producirán seis renglones que contienen los valores 1 1, 1 2, 1 3, 2 1, 2 2 y 2 3. Obsérvese que las variables de control de los ciclos FOR anidados no pueden ser las mismas.

```
FOR i := 1 TO 2 DO
FOR j := 1 TO 3 DO writeln(i,j)
```

Ejemplo: Determine y exhiba el valor de 1 + 2 + 3 + … + num, donde num es un entero mayor de cero. Se necesitará un contador entero para el ciclo (i) y una variable para almacenar las sumas parciales (suma). Asi, la solucion sera:

```
suma := 0;
FOR i := 1 TO num DO suma := suma + i;
writeln('La suma es', suma:1)
```

Observe que no requiere una pareja BEGIN-END en el ciclo FOR, puesto que el cuerpo del ciclo solo contiene una proposición.

ESTRUCTURAS DE CONTROL DEFINIDAS O INDEFINIDAS

La proposición FOR establece lo que se conoce comúnmente como una **estructura de control definida**, ya que los valores iniciales y finales especificados para la variable de control del ciclo determinan de manera exacta el número de ejecuciones del cuerpo del ciclo. Al diseñar la solución de un problema conviene tener presentes las limitaciones del ciclo FOR:
-Es preciso determinar el número de veces que se va a ejecutar el cuerpo del ciclo antes de iniciar su ejecución.
-No es posible terminar "antes de tiempo" ;la ejecución de un ciclo FOR.
Por ejemplo, si se desea calcular el ingreso promedio de un grupo de 50 personas, es apropiado utilizar una proposición FOR, ya que se debe ejecutar exactamente 50 veces el cuerpo del ciclo. En cambio, si no se conoce el número de personas que constituyen el grupo, será preciso utilizar una **estructura de control indefinida**, como la que se puede establecer mediante las proposiciones WHILE y REPEAT-UNTIL. Las estructuras de control indefinidas se pueden usar cuando el ciclo tiene las siguientes características:
-No se sabe necesariamente antes de la ejecución del ciclo el número de veces que se debe ejecutar el cuerpo del ciclo.
-Se puede utilizar más de una condición para terminar la ejecución del ciclo.
Recuérdese que el ciclo REPEAT-UNTIL siempre ejecuta el cuerpo del ciclo por lo menos una vez. Es pertinente indicar que la proposición FOR no se debe usar cuando quizá se emplee más de una condición para terminar el ciclo.
Pascal cuenta con una proposición llamada **proposición de transferencia incondicional** que permite la continuación en un punto razonablemente arbitrario del programa para hacer que termine el ciclo y continúa con la ejecución en otro punto. Esta estrategia definitivamente no se recomienda.

La **ejecución secuencial** es la ejecución de una secuencia de proposiciones, una después de otra, se ejecuta la proposición 1 y después la proposición 2 y así sucesivamente, hasta ejecutarse la última.
La **selección** permite escoger diferentes cursos de acción de acuerdo con el valor de una expresión. La selección se realiza en Pascal mediante las proposiciones IF-THEN-ELSE, IF-THEN y CASE.
Las **estructuras de control cíclicas** permiten especificar la repetición controlada de una o más proposiciones. La repetición se puede controlar por medio de una expresión booleana (cuyo valor indica cuándo se debe continuar la repetición del cuerpo de un ciclo WHILE o REPEAT-UNTIL) o mediante una escala de valores ordinales (en una proposición FOR). Pascal

cuenta con dos estructuras de control de ciclos indefinidos (los ciclos WHILE y REPEAT) y una estructura de control de ciclos definida (el ciclo FOR).

CICLOS INFINITOS

Si el programa se está ejecutando y no termina, es posible que esté en un ciclo infinito. La siguiente es una lista de los puntos que se podrían revisar:

-Asegurarse de que se asignó un valor inicial a la variable de control antes de comenzar cada ciclo WHILE.

-Asegurarse de que se modifica la variable de control dentro del cuerpo del ciclo.

-Verificar que no siempre se cumpla la expresión booleana. Por ejemplo, la proposición WHILE (1 + 1 = 2) DO siempre provocará un ciclo infinito.

-Verificar que no haya una proposición vacía inmediatamente después del DO. Por ejemplo WHILE (num >10)DO; es un ciclo infinito, ya que la variable de control num nunca cambia. Si se coloca por equivocación un signo de punto y coma inmediatamente después del DO, el compilador de Pascal no informa de un error de sintaxis. Se supondrá que la proposición vacía es la proposición que se desea ejecutar repetidamente.

-En las proposiciones REPEAT-UNTIL es preciso asegurarse de que la expresión booleana no siempre es falsa.

-Al escribir ciclos, es conveniente comprobar que se entra al ciclo en las condiciones apropiadas. No hay que olvidar la revisión de los valores iniciales y finales. Estos se llaman **condiciones de frontera**. Muchos errores de los ciclos se deben a que el valor inicial es incorrecto o bien a que nunca se llega al valor final.

capítulo 7

PROCEDIMIENTOS Y FUNCIONES

Los procedimientos permiten declarar variables de manera local, es decir dentro de ellos.
Un ejemplo de procedimiento es:

```
PROCEDURE circulo (radio:real; VAR area,circun: real);
(* Procedimiento para calcular el area y *)
(* circunferencia de un circulo, dado su radio *)

CONST
  pi = 3.14159;

BEGIN
  area: pi * sqr (radio);
  circun := 2 * pi * radio
END;
```

Nótese que se usó la función estándar (incluida) sqr de Pascal para obtener el cuadrado del

radio en el cálculo del área. Recuerde que la palabra reservada VAR no precede al parámetro de entrada (de valor) radio.

PARÁMETROS DE VALOR Y VARIABLES

El encabezado del procedimiento contiene el nombre del procedimiento y una lista opcional de nombres de variables con tipos de datos. Los nombres de variables en el encabezado del procedimiento se llaman **parámetros formales**, ya que no se conocen los valores verdaderos en tanto no se llame o invoque al procedimiento. Todos los parámetros formales deben incluir un tipo de datos en el encabezado y algunos van precedidos de la palabra reservada VAR. Cierta información, como radio en el procedimiento círculo, se proporciona al procedimiento como datos de entrada y no debe cambiarse en el programa que llama. Los parámetros formales que corresponden a los datos de entrada de un procedimiento se llaman parámetros de entrada (de valor). En Pascal se le llama sencillamente **parámetro de valor.** En el encabezado del procedimiento, la lista de parámetros de valor no incluye la palabra reservada VAR.
Los parámetros formales que corresponden a la salida de procedimiento, como son el área y la circunferencia del círculo, se conocen como parámetros de salida (variables). En Pascal se les llama **parámetros de variables**. Cuando se escribe la lista de estos parámetros es preciso emplear la palabra reservada VAR. Esto garantiza que los parámetros verdaderos en el programa que llama van a cambiar. Por ejemplo, si se invoca el procedimiento círculo mediante la siguiente proposición que incluye los parámetros formales

círculo (5,área,circunferencia)

El procedimiento modificará el contenido de los parámetros variables área y circunferencia. El parámetro de valor 5 no cambiará. Cuando se invoca un procedimiento, el orden y tipo de datos de los parámetros verdaderos deben concordar con el orden y tipo de datos de los parámetros formales. El parámetro verdadero que corresponde a un parámetro de valor puede ser una variable, una constante o una expresión. Cuando se invoca el procedimiento, se obtiene el valor del parámetro de valor y se pasa al procedimiento. Por ejemplo, la siguiente invocación de procedimiento pasa una expresión aritmética como parámetro de valor verdadero:

circulo (2 * 3 - 1, área, circunferencia)

Cuando se invoca el procedimiento, el parámetro de valor formal radio tendrá el valor calculado de cinco. Por otro lado, el parámetro verdadero que corresponde a un parámetro variable debe ser una variable. Por tanto, la invocación del procedimiento

círculo (5,2,1)

no es válida, ya que los parámetros segundo y tercero son parámetros de variable y 2 y 1 no son variables. Algunos parámetros pueden funcionar como entrada y salida simultáneamente.

En Pascal, a estos se les llama también parámetros variables y deben incluir la palabra reservada VAR en la lista de parámetros formales del encabezado del procedimiento.
En general, si el procedimiento envía información de vuelta al programa que lo llamó, se deben usar parámetros variables. Por otro lado, si la información se pasa al procedimiento únicamente como datos de entrada, deben usarse parámetros de valor. Existen algunas excepciones a esta regla. Pero, ¿por qué no usar sólo parámetros variables? básicamente, la razón es que cuando se usa un parámetro de valor, se hace una copia del valor y el procedimiento emplea esta copia. Esto garantiza que el procedimiento no puede cambiar el parámetro de valor original. No obstante, los parámetros de valor pueden ser costosos. Es posible que se quiera pasar como datos de entrada a un procedimiento un conjunto de muchos valores. Aun cuando estos valores no funcionen sino como datos de entrada normalmente conviene declararlos como parámetros variables. Si no se hace así, la computadora gasta tiempo y localidades de memoria al copiar el conjunto de valores. Debe tenerse cuidado de que el procedimiento no cambie inadvertidamente estos valores. He aquí un resumen de algunos puntos importantes en lo que concierne a los parámetros de valor y variables.

- Se debe incluir una lista de los parámetros formales en el encabezado del procedimiento.
- La proposición de invocación del procedimiento pasa los parámetros verdaderos.
- El orden y tipo de datos de los parámetros formales deben ser los mismos que para los parámetros verdaderos.
- Los parámetros variables deben incluir la palabra reservada VAR en la lista de parámetros de un encabezado de procedimiento.
- El parámetro verdadero de un parámetro variable debe ser una variable.
- El parámetro verdadero de un parámetro de valor puede ser una variable, constante o expresión.

Procedimientos anidados
Cuando se escriben procedimientos dentro de otros procedimientos se dice que están **anidados**. Ejemplo:

```
PROCEDURE dólar
(* Lee hasta encontrar $ *)
VAR
  uncar:char;
BEGIN
  REPEAT
    read(uncar)
  UNTIL (uncar = ´$´);
END;
```

Obsérvese que se declaró una variable de carácter llamada uncar dentro del procedimiento para almacenar el carácter leído. Esta variable es una variable local que se conoce únicamente en el procedimiento dólar. Este procedimiento no tiene parámetros formales porque ya se sabe

que el último carácter leído será un signo de dólar.

```
PROCEDURE sueldo (limite:integer;VAR salario:integer; VAR indic:boolean);
(* Lee el salario y determina si rebasa el límite *)
BEGIN
  read(salario);
  indic:=salario > límite
END;
```

Obsérvese que en el encabezado del procedimiento aparece dos veces la palabra VAR. Si se omitiera el último VAR (antes de indic), indic sería un parámetro de valor y la proposición de asignación

```
indic := salario > límite
```

no cambiaría al parámetro verdadero. Cabe hacer notar, empero, que en este caso indic cambiaría localmente.

Reglas de alcance

Cuando se tienen procedimientos anidados, el programador debe ocuparse de variables locales y globales y su alcance. El **alcance** de un identificador es aquella parte del programa en la que se conoce el identificador. Puesto que cualquier procedimiento puede incluir declaraciones CONST y VAR, los identificadores se conocen como constantes locales o variables locales en el procedimiento. Cada uno de los identificadores se conoce en todos los segmentos del procedimiento, incluso en los procedimientos anidados que no tienen un identificador con el mismo nombre.

Aun cuando es posible hacer referencia a una variable global desde un procedimiento dentro del alcance de la variable, generalmente no es buena idea. Es preferible pasar la variable como parámetro. No conviene cambiar las variables globales mediante una proposición de asignación directa dentro del procedimiento (este es un ejemplo de **efectos secundarios**). Los procedimientos deben ser módulos autosuficientes e independientes. Los efectos secundarios muchas veces pueden provocar problemas serios.

FUNCIONES

Las **funciones** son subprogramas que calculan un solo valor. Por ejemplo, la función estándar sqrt calcula la raíz cuadrada de un número no negativo.

Funciones estándar

Las funciones estándar son funciones predefinidas con que cuenta Pascal estándar. Ya se vieron las siguientes funciones aritméticas: sqrt, sqr, abs, round, trunc. Las funciones aritméticas que faltan son las funciones trigonométricas **sin**,**cos** y **arctan** y las funciones exponencial y logaritmos natural **exp** y **ln**.

Las funciones estándar también incluyen las cuatros funciones ordinales **pred**, **succ**, **ord**, **chr**. Las demás funciones estándar son aquellas que producen resultados booleanos.

Específicamente, estas funciones son:
odd Determina si su argumento es un entero impar
eoln Determina si el siguiente carácter en los datos de entrada es un carácter de fin de línea
eof Determina si el siguiente carácter en los datos de entrada es un carácter de fin de archivo

Las funciones que producen resultados booleanos se llaman en ocasiones **predicados**.

Función odd
La función **odd** produce el resultado true si el argumento entero es un número impar y produce el resultado false si el argumento es un entero par.
Un ejemplo de esto:

```
read(numero);
If odd(numero) Then
  WriteLn('Impar');
Else
  WriteLn('Par');
```

También se puede usar la función odd para determinar si un número es par:

```
If Not odd(numero) Then…
```

Funciones eoln y eof
Los datos que se introducen cuando se está ejecutando el programa y los datos que se introducen en un archivo mediante un editor de textos deben terminar de tal manera que un programa que lea los datos pueda determinar cuándo ha llegado al final. Cuando se emplea un programa en Pascal para crear un archivo de salida, el centinela se agrega automáticamente. Este centinela se llama **fin de archivo** (end-of-file), y la función estándar que detecta el hecho de haberlo encontrado se llama **eof**. Los sistemas ponen un límite a la longitud máxima que deben aparecer como un solo renglón de caracteres en la terminal. Por tanto, a todas las líneas se agrega un centinela especial conocido como carácter de **fin de línea**, el cual se representa como **eoln**. La función **eoln** permite al programador en Pascal determinar si el siguiente carácter en la entrada es el fin de línea. Un ejemplo de esto:

```
Juan Barrera $40000<eoln>
Sandra frias  $45000<eoln>
Delia Perez   $35000<eoln>
<eof>
```

Las funciones eoln y eof producen el resultado true cuando el siguiente carácter del archivo es <eoln> o <eof>. De manera similar, cuando el siguiente carácter del archivo no es <eoln> o <eof>, las funciones producen el resultado false.

Cuando se invoca la función eoln se examina el siguiente carácter del archivo (es decir, el que sigue al último carácter leído) sin verlo realmente. Para usar la función eoln no se lee un carácter y se le examina después para determinar si es el carácter de fin de línea, sino que se pregunta si el siguiente carácter de los datos de entrada es el fin de línea.
Ejemplo:

```
While Not eoln Do (* Mientras no se llegue al fin de línea *)
Begin
  Read(uncar);     (* Capturar el siguiente elemento *)
  Write(uncar);     (* Exhibirlo *)
End;
WriteLn;   (* Exhibir un carácter de fin de línea *)
ReadLn    (* y pasar por alto el fin de línea en la entrada *)
```

Es preciso hacer hincapié en un aspecto importante de este ejemplo. Cuando este ciclo termina, el carácter de fin de línea no se habrá leído. Se debe mencionar lo siguiente acerca del procesamiento de <eoln>:
-La función eoln prueba si existe el carácter <eoln>, pero no lo lee.
-La proposición Write jamás puede escribir un carácter <eoln>

La razón de que no se pueda leer simplemente el <eoln> asignando a uncar para producir un fin de línea en la salida es que el <eoln> se transforma en un espacio en blanco (‘ ‘) cuando se lee. La única forma de detectar un carácter de fin de línea es la función eoln.
La función eof funciona de manera similar, sirve para determinar si el siguiente carácter del archivo de entrada es el carácter de fin de archivo; nunca se lee realmente. Cualquier intento de leer el carácter eof producirá un mensaje de error. Un ejemplo:

```
While Not eof Do (* Mientras no se llegue al fin de archivo *)
Begin
  While Not eoln Do (* Mientras no se llegue al fin de línea *)
  Begin
    Read(uncar);     (* capturar el siguiente carácter *)
    Write(uncar);      (* exhibirlo *);
  End;
  WriteLn;          (* exhibir un carácter de fin de línea *)
  ReadLn;         (* y pasar por alto el fin de línea de la entrada *)
End;
```

Una vez más se recalca que el fin de archivo no se debe leer. Es por esta razón que se utilizó un ciclo While en vez de un ciclo Repeat.

FUNCIONES DEFINIDAS POR EL USUARIO

Las **funciones definidas por el usuario** en Pascal son similares a los procedimientos, con la excepción de que el objetivo de una función es devolver únicamente un valor al punto en que se le llamó. Por ejemplo, suponga que se desea escribir una función llamada cuarta que produce la cuarta potencia de un número del tipo real. Es decir, si el número se llama num, se desea calcular:

num^4 = num * num * num * num

La siguiente función en Pascal realiza este cálculo:

```
Function cuarta(número:real):real;
(* Calcular la cuarta potencia del argumento *)
Begin
  cuarta := sqr(sqr(numero));
End;
```

Obsérvese que el encabezado de la función se parece al encabezado de un procedimiento. La palabra reservada **FUNCTION** va seguida del identificador de la función y la lista de parámetros. Esta va seguida de un signo de dos puntos y el identificador de tipo que especifica el tipo de valor que va a tener el resultado de la función. En el caso de cuarta, el resultado será un valor real.

Identificador -> (Lista de parámetros): -> Identificador de tipo ;

El mecanismo que siguen las funciones en Pascal para especificar el valor que se devolverá es asignar el valor al nombre de la función. En la función cuarta la proposición de asignación

```
cuarta:= sqr(sqr(numero))
```

Se asigna la cuarta potencia de número al nombre de la función con lo que se especifica que este valor es el que se debe devolver como valor de la función cuando esta termine. Se pueden hacer varias asignaciones al nombre de la función, pero solo se devuelve el último valor asignado. Si el cálculo requiere la devolución de más de un valor, es preciso emplear un procedimiento en vez de una función. Dado que las funciones proporcionan únicamente un resultado, y para hacer hincapié en el parecido entre las funciones matemáticas y las funciones de Pascal definidas por el usuario, por lo regular no se utilizan parámetros variables con las funciones.

Es importante no confundir el nombre de la función con una variable. Usar el nombre de la función en una expresión (por ejemplo: en el lado derecho de una proposición de asignación) no proporcionará el último valor asignado al nombre de función, sino más bien especificará que se debe invocar la función para producir un valor como resultado de la invocación.

Un ejemplo de una función que evalúa si dos números son pares o impares y devuelve true, en caso contrario devuelve false.

```
Function paridad(num1,num2:integer): Boolean;
(* Resulta true si num1 y num2 son ambos pares *)
(* o ambos impares, y false si no es así. *)
Begin
  paridad:= (odd(num1)) And (odd(num2)) Or (Not odd(num1) And Not odd(num2))
End;
```

Una vez más, el tipo de dato del resultado debe seguir la lista de parámetros y el signo de dos puntos. La función contiene una proposición de asignación con el nombre de la función en el lado izquierdo de la asignación. El valor resultante siempre se asigna al nombre de la función. La siguiente función calcula la tangente de un ángulo.

```
Function tan(x:real):real;
Begin
  tan:= sen(x) / cos(x)
End;
```

Otro ejemplo de función para calcular a^b, mediante la propiedad que dice que a^b=e^(b ln a)

```
Function potencia(a,b:real):real;
Begin
  potencia := exp(b * ln(a))
End;
```

Invocación de funciones

Al igual que en el caso de los procedimientos, para invocar una función se requiere que el orden y tipo de datos de los parámetros verdaderos concuerden con el orden y tipo de datos de los parámetros formales. Si esto no se cumple, el compilador lo detectará e informará al usuario del error.

La invocación de funciones no es igual a la invocación de procedimientos. La razón de esta diferencia es que la función devuelve un valor al punto de invocación, aunque no a través de los parámetros mismos, como en el caso de los procedimientos. A resultas de esto, para invocar las funciones definidas por el usuario se escribe su nombre y lista de parámetros en una expresión, al igual que con las funciones estándar. Por ejemplo, para determinar el valor de dos a la tercera potencia y almacenar este valor en la variable x, se escribirá:

```
x:= potencia(2.0,3.0)
```

y para calcular y exhibir la tangente de 17.3 se escribiría

```
WriteLn (‘Tangente de 17.3 = ’, tan(17.3))
```

sin olvidar que una proposición Write y WriteLn puede exhibir el valor de una expresión.
Las invocaciones de funciones se tratan igual que cualquier otro valor del tipo que produce la función. Hasta es posible combinar funciones. Por ejemplo, para calcular la tangente de x elevada a la tercera potencia y después asignar la mitad de este valor a y, se escribiría.

```
y:= potencia(tan(x),3.0)/2.0
```

El nombre de la función debe aparecer en el lado izquierdo de por lo menos una proposición de asignación dentro de la función misma para comunicar el resultado de la función al punto de invocación. Por lo general, el nombre de la función no deberá aparecer en el lado derecho de una proposición de asignación dentro de la función misma.
Un mal uso de función es el siguiente, puesto que aparece el nombre de la función al lado derecho de la proposición de asignación.

```
Function potencia10 (x:integer):integer;
(* función que no funciona *)
Var
  i:integer;
Begin
  potencia10:= 1;  (* asignar el valor inicial al resultado *)
  For i:=1 To 10 Do
    potencia:= potencia10 * x
End;
```

Para modificar la función se puede agregar una variable temporal con el fin de guardar el resultado parcial y después una proposición para asignar el resultado final al nombre de la función, he aquí la solución correcta.

```
Function potencia10(x:integer):integer;
(* devolver x a la décima potencia *)
Var
  i,temp:integer;
Begin
  temp:=1; (* asignar valor inicial al resultado *)
  For i:=1 To 10 Do
    temp:= temp * x;
  potencia:=temp;
End;
```

Cuando aparece el nombre de la función en una expresión, se invoca la función. Esto se cumple aun en el caso de expresiones dentro de la función. Las funciones que se invocan a sí misma se llaman **funciones recursivas**.

Procedimientos y funciones recursivas

Una función o procedimiento que se llama o invoca a sí mismo se dice que es **recursivo**. Las funciones recursivas son apropiadas cuando se puede definir un problema en términos de sí mismo (de manera recursiva), cuando la computadora encuentra una llamada de función recursiva, debe postergar temporalmente el cálculo para evaluar la función recursiva. Una vez que se obtiene el resultado de la llamada recursiva, continua el cálculo de la función. Aunque existe la posibilidad de utilizar funciones recursivas es una mejor práctica utilizar la iteración con una variable asignada localmente al procedimiento o función.

Recordatorios de Pascal

- Los procedimientos o funciones se deben declarar físicamente antes de invocarlos.
-Los parámetros formales y verdaderos deben tener el mismo orden y tipo de datos.
-El parámetro verdadero que corresponda a un parámetro VAR formal debe ser una variable.
-Los parámetros de valor pueden ser variables, constantes o expresiones.
-Los parámetros de valor sirven como datos de entrada para un procedimiento o función.
-La palabra reservada VAR debe preceder al parámetro variable para cada tipo de datos en la lista de parámetros formales:
Procedure ej (VAR n:real; VAR car1, car:char);
-No conviene hacer referencia a las variables globales directamente desde el interior de un procedimiento o función.
-Es conveniente que las variables y constantes que se utilizan únicamente en un procedimiento o función sean variables locales.
-Un identificador local que tiene el mismo nombre que un identificador global tienen prioridad dentro del procedimiento o función.
-No se puede hacer referencia a los identificadores locales fuera de su alcance.
-La función booleana eoln produce el resultado true si el siguiente carácter que se va a leer es el carácter de fin de línea (<eoln>).
-La función booleana eof produce el resultado true si el siguiente carácter es el fin de archivo (<eof>).
-El tipo de datos del resultado de la función debe incluirse en el encabezado de la función:
Function potencia(base,exponente:real):real;
-El nombre de la función debe aparecer en el lado izquierdo de por lo menos una proposición de asignación de la función.
-El nombre de la función no debe aparecer en una expresión dentro de la función.
-Las invocaciones de fusión no son válidas cuando se escriben como invocaciones de procedimientos. Deben aparecer como componentes de expresiones:
potencia (2.0, 3.0) (* no válida *)
x:= potencia (2.0, 3.0) (* válida *)
-Las funciones se usan cuando se desea obtener únicamente un valor, en otros casos se usa un procedimiento.

Diferencia clave entre Parámetros Nominales y Efectivos

Parámetros Nominales: Son las "etiquetas" o nombres que se utilizan en la definición de una función o procedimiento. Se definen cuando se declara la función o procedimiento y se utilizan dentro de él para referirse a los datos que se pasarán.

Parámetros Efectivos: Son los valores reales o las variables que se pasan a la función o procedimiento cuando se invoca. Estos valores o variables se asignan a los parámetros nominales en el momento de la llamada a la función o procedimiento.

Relación entre Parámetros Nominales y Efectivos: Cuando se llama a una función o procedimiento, los valores de los parámetros efectivos se asignan a los parámetros nominales. Durante la ejecución de la función o procedimiento, se trabaja con los parámetros nominales, que internamente contienen los valores de los parámetros efectivos.

Relación entre los Tipos de Parámetros y el Paso de Parámetros:

- **Parámetros de Entrada:** Usualmente se pasan **por valor** porque no se espera que se modifiquen dentro de la función o procedimiento. Se utilizan para proporcionar datos a la función.
- **Parámetros de Salida:** Suelen pasarse **por referencia** (usando `var`) porque el propósito es modificar el valor del parámetro dentro de la función o procedimiento y reflejar ese cambio fuera de él.
- **Parámetros de Entrada/Salida:** En algunos casos, un parámetro puede servir como entrada y salida al mismo tiempo, es decir, se le pasa un valor que puede ser leído y modificado dentro de la función o procedimiento. En estos casos, el parámetro también se pasa por referencia.

Parámetro por Valor

Cuando un parámetro se pasa por valor, se envía una copia del valor de la variable al procedimiento o función. Esto significa que cualquier modificación realizada en el parámetro dentro del procedimiento no afecta a la variable original en el ámbito del llamado.

Parámetro por Referencia

Cuando un parámetro se pasa por referencia, se pasa una referencia a la variable original en lugar de una copia de su valor. Esto significa que cualquier modificación realizada en el parámetro dentro del procedimiento o función afectará directamente a la variable original.

Los procedimientos se pueden clasificar en **procedimientos de entrada**, **procedimientos de salida** y **procedimientos internos** basados en el tipo de operación que realizan y en cómo

interactúan con el programa. Aquí te explico cada tipo en detalle:

1. Procedimientos de Entrada

Procedimientos de entrada se encargan de recibir datos del usuario o de una fuente externa y procesarlos para su uso en el programa. Su función principal es capturar información que el programa necesita para operar. No tienen parámetros por referencia.

Ejemplo de procedimientos de entrada:

- **Leer datos del usuario.**

2. Procedimientos de Salida

Procedimientos de salida se encargan de mostrar datos al usuario o enviar datos a una salida externa, como una pantalla o un archivo. Su función principal es presentar información procesada o resultados al usuario o guardar datos para su uso posterior. Sus parámetros son por referencia.

Ejemplo de procedimientos de salida:

- **Mostrar datos en la pantalla.**

3. Procedimientos Internos

Procedimientos internos son procedimientos que realizan operaciones dentro del programa, pero no están directamente relacionados con la entrada o salida de datos. A menudo, estos procedimientos manejan la lógica interna, cálculos, y otras operaciones que no implican interacción directa con el usuario o archivos externos. Contienen los dos tipos de parámetro o cualquiera de los dos por sí solo.

Ejemplo de procedimientos internos:

- **Realizar cálculos.**

El acceso a procedimientos anidados está restringido al procedimiento que los contiene. Esto es parte del diseño del ámbito en Free Pascal y otros lenguajes similares, y asegura que los procedimientos anidados sólo puedan ser utilizados dentro del contexto de su procedimiento padre.