Capítulo 2

Algoritmos, Pseudocódigos y Programación Estructurada

En el Capítulo 1: Introducción a la Programación, se mencionó que la computadora sólo entiende una serie de instrucciones codificadas en el lenguaje máquina de la propia computadora. En este lenguaje, las instrucciones son una secuencia de ceros y unos (código binario) y escribir un programa directamente en este lenguaje sería una tarea laboriosa y tediosa. También es seguro que el proceso de detectar errores en el código y su corrección sería un proceso que nos tomaría aún más tiempo y nos produciría un buen dolor de cabeza.

Afortunadamente, los primeros programadores al enfrentarse a los problemas ya mencionados, buscaron la forma de reducir el esfuerzo que implicaba escribir programas. La solución que encontraron fue la de crear lenguajes, llamados de alto nivel, cuya sintaxis fuese más parecida a la del lenguaje humano permitiéndole al programador escribir programas en un lenguaje que se entiende directamente. Es la misma computadora la que se encargara de traducir los programas en el código de alto nivel al código máquina mediante unos programas llamados traductores.

Procedimiento Para Crear un Programa

El procedimiento para crear un programa que la computadora pueda ejecutar, es el siguiente:

1. **Comprender el problema** al que se le está buscando solución y especificar qué información se le proporcionará al programa, **datos de entrada** y qué tipo de información arrojará el programa como respuesta, **datos de salida**.

2. **Diseñar el algoritmo** del programa que convertirá los datos de entrada al programa en los datos de salida del programa.

3. **Codificar el algoritmo** en un lenguaje de alto nivel.

4. Usar un editor de textos para **editar el código**. Este código se conoce como **programa fuente**.

5. Convertir el programa fuente a **código máquina** usando un compilador. Este proceso se llama **compilar el programa**. Normalmente nuestro código fuente tendrá **errores de sintaxis** que el compilador detectará y nos lo indicará. Deberemos regresar al editor para corregirlos y repetir el paso 5 hasta que nuestro código no tenga errores de sintaxis. El código máquina resultante, también llamado **código ejecutable** es un programa que la máquina puede ejecutar.

6. **Ejecutar** el programa alimentándolo con datos para los cuales conozcamos los resultados que nos arrojará el programa y **comparar los resultados obtenidos del programa con los resultados esperados**. Es frecuente que los resultados obtenidos no coincidan con los esperados. Esto se debe a que nuestro algoritmo contiene algún **error de lógica** que deberá de encontrarse y corregirse.

En este tema estudiaremos los primeros dos pasos del procedimiento para crear un programa Los restantes se verán en el los tutoriales del sistema de desarrollo empleado en el curso.

Comprensión del Problema

A fin de que podamos resolver cualquier problema, en cualquier disciplina no sólo en computación, es necesario comprender el problema a resolver. Es muy frecuente que comencemos a plantear soluciones antes de tener en claro el problema a resolver. Esto nos trae como consecuencia que los resultados obtenidos sean incorrectos y que tengamos que buscar una nueva solución. Además hay que considerar que en la práctica, los problemas a los que se les busca solución mediante la computadora son problemas complejos y que por lo general requieren del trabajo de varias personas, por períodos de tiempo largo. De aquí la importancia que tiene entender con claridad el problema antes de dedicarle recursos a resolverlo.

Una parte fundamental de la comprensión de un problema es la identificar correctamente qué respuestas son las que deseamos que nos proporcione el programa y cuál es la información que necesitamos proporcionarle al programa.

Algoritmo y Pseudocódigo

Una vez entendido el problema, el siguiente punto es establecer la secuencia de pasos necesarios para obtener las respuestas deseadas a partir de esos datos. Esta secuencia de pasos escrita en manera formal y sistemática se conoce como **algoritmo**.

Para describir este algoritmo se requiere de un lenguaje. Este lenguaje debe permitirnos, aparte de la descripción de los pasos para resolver el problema, modelar la representación de la solución. Esto último significa que a partir de la descripción de la solución, otra persona además de la que escribió el algoritmo sea capaz de llegar a la misma solución. Tal lenguaje se llama **pseudocódigo**.

Un algoritmo escrito en pseudocódigo, es un conjunto de **sentencias** escritas siguiendo cierta sintaxis o reglas de construcción.

Sentencias

Las sentencias, llamadas también **estructuras de control**, son construcciones para dirigir el flujo de acciones que la computadora efectuará sobre los datos para obtener los resultados.

Hay tres tipos de sentencias o estructuras de control: La sentencia secuencial o compuesta, la sentencia condicional o selección y la sentencia repetitiva o iteración.

Sentencia Compuesta

Una característica fundamental de un algoritmo, es que está formado por un conjunto de instrucciones o pasos que se ejecutan en una secuencia bien definida. A este conjunto de instrucciones que se ejecutan en secuencia se le conoce como **sentencia compuesta**. Cada una de las instrucciones de una sentencia compuesta puede ser una sentencia simple, una sentencia compuesta, una sentencia condicional o una sentencia repetitiva. Una **sentencia simple**, es una instrucción que no puede descomponerse en instrucciones más sencillas.

En pseudocódigo, una sentencia compuesta se escribe escribiendo cada una de sus sentencias que la componen en un renglón por separado:

```
sentencia1
[sentencia2]...
```

Note que una sentencia compuesta puede estar formada de una o más sentencias.

Ejemplo sobre la Sentencia Compuesta

Considere el siguiente problema:

> Escribir el pseudocódigo para un programa que convierta una velocidad dada en km/hr a m/s.

Podemos ver que la respuesta esperada de este programa es un número que representa la velocidad expresada en m/s. Además, el programa tiene como dato de entrada un número que representa la velocidad expresada en km/hr. El factor de conversión no es un dato de entrada puesto que su valor no cambia y puede considerarse como una constante en el programa.

La búsqueda de la solución de este problema se hará en varias aproximaciones. Empezaremos con una solución muy general y la iremos refinando en aproximaciones sucesivas. A esta técnica se le conoce con el nombre de **diseño descendente**. En la primera aproximación se establece lo **que** hay que hacer para resolver el problema sin preocuparnos en el **cómo**. En las siguientes aproximaciones se refinará la solución detallando el cómo. La primera aproximación de la solución podría ser el siguiente pseudocódigo:

```
lee velocidad
convierte velocidad de km/hr a m/s
escribe velocidad
```

La primera sentencia hace que la computadora lea un número suministrado por el usuario del programa y que representa la velocidad en km/hr. Ese número se almacena en una localidad de la memoria RAM llamada **variable**. Cada variable tiene un nombre. La variable utilizada en este programa se llama `velocidad`. La segunda sentencia toma el valor de la velocidad en km/hr y los convierte a m/s, dejando el resultado en la misma variable `velocidad` y por último la última instrucción toma el valor de la velocidad en m/s almacenado en la variable `velocidad` y lo escribe para que el usuario conozca el resultado.

Note que las tres sentencias del pseudocódigo deben de ejecutarse en el orden en que fueron escritas. La computadora no puede hacer la conversión de velocidad hasta no conocer la velocidad en km/hr. Tampoco puede escribir la velocidad en m/s antes de haber calculado esta velocidad. Estas tres sentencias constituyen una sentencia compuesta.

En las siguientes aproximaciones de la solución, se refinará cada uno de los pasos de la primera aproximación tratando de expresar el cómo se deben realizar cada uno de esos pasos. Al final esos pasos deberán quedar expresados en términos de las operaciones que es capaz de realizar la computadora, que son ciertas operaciones aritméticas y lógicas. También, ya se ha mencionado que la computadora posee ciertos dispositivos de entrada y salida por medio de los cuales le podemos alimentar o extraer información. Por ello, podemos pensar que la computadora puede “leer” y “escribir” un dato y que deben de existir instrucciones que le indiquen a la computadora que realice esas funciones. En pseudocódigo esas instrucciones se denotan por las **palabras clave**: **`lee`** y **`escribe`.** Podemos pensar que `lee` y **`escribe`** son sentencias simples. La sentencia **`lee`**, permite que el usuario suministre un dato a través de un dispositivo de entrada, normalmente el teclado, y lo almacena en una variable. La sentencia **`escribe`** despliega un resultado en un dispositivo de salida, normalmente el monitor.

Nos resta por analizar la segunda sentencia de nuestro pseudocódigo: Convierte la velocidad de km/hr a m/s. Aquí nos podemos preguntar: ¿Existe una instrucción para efectuar la conversión de una velocidad en km/hr a m/s? La respuesta es no. La tarea de convertir velocidades, así como cualquier otra conversión, puede expresarse a partir de

otras tareas más elementales. La tarea de convertir una velocidad en km/hr a m/s puede escribirse como:

```
toma el dato almacenado en la variable velocidad
multiplícalo por 1000
divide el resultado anterior entre 3600
almacena el resultado en la variable velocidad
```

Los cuatro pasos anteriores pueden representarse en forma abreviada mediante la siguiente expresión:

```
velocidad = velocidad * 1000/3600
```

Donde el símbolo **=** llamado operador de asignación, significa que el valor original de la variable velocidad va a ser reemplazado por el resultado de:

```
velocidad * 1000/3600
```

Esto es, el resultado de multiplicar (*) el valor que está almacenado en la variable velocidad por 1000 y dividirlo (/) entre 3600. Luego una segunda aproximación al programa deseado estaría dada por el siguiente pseudocódigo:

```
lee velocidad
velocidad = velocidad * 1000/3600
escribe velocidad
```

En este programa sólo fueron necesarias dos aproximaciones a fin de que todos los pasos del programa fuesen instrucciones que la computadora pueda ejecutar. En programas más complejos será necesario un mayor número de aproximaciones.

Ejercicio sobre la Sentencia Compuesta

Escribir el pseudocódigo para un programa que lea los tres lados de un triángulo y escriba su área.

Sentencia Condicional

Una segunda sentencia, la sentencia condicional, le permite a la computadora seleccionar entre dos cursos alternativos a seguir, dependiendo de una condición dada. La sintaxis de la sentencia condicional es:

```
si(expresión)
   sentencia1
[otro
   sentencia2]
```

si es una palabra clave con la que inicia la sentencia condicional. La palabra clave **otro** separa las dos sentencias compuestas: *sentencia1* y *sentencia2*. La expresión entre paréntesis al evaluarse sólo puede tomar los valores falso o verdadero. Si el valor de

expresión es verdadero, la computadora ejecuta la *sentencia1* en caso contrario, la computadora ejecutará la *sentencia2*.

Note que la construcción:

```
[otro
   sentencia2]
```

es opcional (está encerrada entre corchetes). Si se omite y el valor de *expresión* es falso, entonces la computadora no hará nada.

Ya mencionamos que una sentencia compuesta puede estar formada de una o más sentencias. Si las sentencias compuestas de la sentencia condicional sólo tienen una sentencia pueden escribirse también de la siguiente manera:

```
si(expresión) sentencia1
[otro sentencia2]
```

Si las sentencias compuestas tienen más de una sentencia, deberán de escribirse como:

```
si(expresión) {
   sentencia11
   sentencia12 ...
}
[otro {
   sentencia21
   sentencia22 ...
}]
```

Note que las sentencias que forman *sentencia1* se han delimitado por llaves, **{}**. Esto nos indica que todas las sentencias se ejecutarán si *expresión* es verdadera. Si omitimos las llaves y escribimos:

```
si(expresión)
   sentencia11
   sentencia12 ...
[otro {
   sentencia21
   sentencia22 ...
}]
```

la construcción es incorrecta, debido a que si *expresión* es verdadera sólo se ejecutaría *sentencia11*. *sentencia12* ... se considerarían como sentencias independientes de la sentencia condicional y la construcción:

```
[otro
   sentencia21
   sentencia22 ...]
```

ya no formaría parte de la sentencia condicional, lo cual no es posible ya que esta construcción no puede existir por sí sola. También las sentencias que forman

sentencia2 se han delimitado por llaves, para indicar que todas las sentencias se ejecutarán si *expresión* es falso. Si omitimos las llaves y escribimos:

```
si(expresión) {
   sentencia11
   sentencia12 ...
}
[otro
   sentencia21
   sentencia22 ...]
```

la construcción aunque correcta, no producirá el efecto deseado ya que si *expresión* es falso sólo se ejecutaría *sentencia21*. *sentencia22* ... se considerarían como sentencias independientes de la sentencia condicional.

Ejemplos sobre la Sentencia Condicional

1. Escribir el pseudocódigo para un programa que lea tres números enteros y escriba el mayor de ellos.

   En este problema, los datos de entrada son tres números positivos, a, b y c y la respuesta esperada es un número, el mayor de los tres. Una primera aproximación a la solución del problema puede ser:

   ```
   lee a, b, c
   encuentra el mayor de a, b y c
   escribe mayor
   ```

   donde a, b y c son las variables que contienen los números de entrada y `mayor` es la variable que contiene el mayor de los tres números. En este pseudocódigo, el único paso que necesita refinarse es el segundo, ya que el primero y el tercero corresponden a sentencias simples. Para encontrar el mayor de a, b y c, la computadora debe comparar los tres números. Sin embargo, la computadora sólo puede comparar dos números a la vez. Por lo que el segundo paso debe separarse en dos partes:

   ```
   encuentra el mayor de a y b y almacénalo en mayor
   encuentra el mayor de c y mayor y almacénalo en mayor
   ```

   Para encontrar el mayor de a y b, la computadora debe preguntarse: ¿es a mayor que b?. Esto lo podemos expresar mediante la sentencia condicional siguiente.

   ```
   si(a > b) mayor = a
   otro mayor = b
   ```

   Aquí la computadora evalúa la expresión a > b. Si es verdadero, esto es, si a > b, le asignará a la variable `mayor` el valor de a, en caso contrario almacenará en `mayor` el valor de b. De cualquier forma, al terminar de ejecutar la sentencia, en mayor tendremos el mayor de a y b.

De la misma manera, para encontrar el mayor de c y mayor, podemos usar la sentencia condicional siguiente.

```
si(c > mayor) mayor = c
```

Si la expresión `c > mayor` es verdadero significa que el mayor de los tres números es c y por lo tanto lo almacenamos en la variable `mayor`. Note que en este caso no hay una sentencia 2 ya que si la expresión `c > mayor` es falso, la variable mayor ya contiene el mayor de los tres números y no es necesario cambiar el valor de esta variable.

El pseudocódigo para este programa queda entonces como:

```
lee a, b, c

si(a > b) mayor = a
otro mayor = b
si(c > mayor) mayor = c

escribe mayor
```

2. Escribe el pseudocódigo de un programa, que realice las cuatro operaciones fundamentales de la aritmética. El programa deberá leer el primer número, el carácter que representa la operación deseada y el segundo número. El programa desplegará el resultado.

   En este programa la entrada estará formada de dos números que representan los operandos y un carácter que representa el operador. La salida consistirá de un número que representa el resultado o un mensaje de error si la operación es inválida.

   La primera aproximación de la solución del problema puede ser:

```
lee resultado
lee operador
lee operando
calcula resultado
escribe resultado
```

   El primer operando se almacena en una variable llamada `resultado` y es en ésta donde queda el resultado al final del programa. La variable `operador` contiene el operador (carácter empleado para indicar la operación) y la variable `operando` contiene el segundo de los operandos. En este pseudocódigo, el único paso que necesita refinarse es el cuarto. Los otros corresponden a sentencias simples de lectura y escritura. La forma en que se calcula el resultado depende de la operación a realizar y está determinada por el contenido de la variable `operador`. El cálculo del resultado se puede escribir mediante las siguientes sentencias condicionales en cascada:

```
si(operador == '+') resultado = resultado + operando
```

```
otro si(operador == '-') resultado = resultado - operando
otro si(operador == '*') resultado = resultado * operando
otro si(operador == '/')
        si(operando != 0.0) resultado = resultado / operando
        otro escribe "Error: División entre cero"
otro escribe "Error: Operador desconocido"
```

En la expresión de la sentencia condicional:

```
operador == '+'
```

el símbolo **==** significa "igual a", por lo que la computadora se pregunta si el carácter almacenado en la variable operador es igual al carácter '+'. Si es cierto se sumarán los contenidos de las variables resultado y operando y el resultado se almacenará en la variable resultado. En caso contrario se investiga si la operación deseada es la resta, etc. Note que si la operación deseada es la división, se compara el valor en la variable operando con cero. Si el valor en operando no es cero podemos efectuar la división, en caso contrario desplegaremos un mensaje de error ya que la división entre cero no es válida.

El pseudocódigo para este programa queda como:

```
lee resultado
lee operador
lee operando

si(operador == '+') resultado = resultado + operando
otro si(operador == '-') resultado = resultado - operando
otro si(operador == '*') resultado = resultado * operando
otro si(operador == '/')
   si(operando != 0.0) resultado = resultado / operando
   otro escribe "Error: División entre cero"
otro escribe "Error: Operador desconocido"

escribe resultado
```

Ejercicios sobre la Sentencia Condicional

1. El costo de un telegrama ordinario es de $1000 si el número de palabras es hasta 10, por cada palabra adicional se cobra $200. Si el telegrama es urgente los costos son de $2000 y $400 respectivamente. Escribir el pseudocódigo para un programa que lea el tipo del telegrama (una sola letra, 'O' para ordinario y 'U' para urgente) y el número de palabras del telegrama y escriba el costo de éste.

2. Escribir el pseudocódigo para un programa que lea tres números positivos suministrados en orden ascendentes los cuales representan las longitudes de los lados de un triángulo. El programa deberá determinar si los tres lados forman un triángulo y su tipo.

Sentencia Repetitiva

La sentencia repetitiva, le permite a la computadora ejecutar una serie de pasos o instrucciones en forma repetitiva mientras se cumpla una condición dada. La sintaxis de esta sentencia es la siguiente:

```
mientras(expresión)
   sentencia
```

La sentencia repetitiva inicia con la palabra clave **mientras.** La expresión entre paréntesis al evaluarse sólo puede tomar los valores falso o verdadero. Primero la computadora evalúa la *expresión.* Si es verdadero, la computadora ejecuta la sentencia compuesta, *sentencia*, enseguida vuelve a evaluar la *expresión* y continuará ejecutando la sentencia compuesta mientras *expresión* sea verdadero. Si valor de *expresión* es falso, entonces la sentencia repetitiva termina.

Al igual que con la sentencia condicional, si la sentencia compuesta de la sentencia repetitiva solo tiene una sentencia puede escribirse como

```
mientras(expresión) sentencia
```

Si la sentencias compuesta tiene más de una sentencia, deberá de escribirse como:

```
mientras(expresión) {
   sentencia1
   sentencia2 ...
}
```

Note que las sentencias que forman *sentencia* se han delimitado por llaves, **{}**. Esto nos indica que todas las sentencias se ejecutarán si *expresión* es verdadero. Si omitimos las llaves y escribimos:

```
mientras(expresión)
   sentencia1
   sentencia2 ...
```

la construcción aunque correcta, no producirá el efecto deseado ya que si *expresión* es verdadero sólo se ejecutaría *sentencia1*. *sentencia2* **...** se considerarían como sentencias independiente de la sentencia repetitiva.

Ejemplos sobre la Sentencia Repetitiva

1. Escribir el pseudocódigo para un programa que genere una tabla que muestre el capital acumulado de una inversión, a una tasa de interés anual, con recapitalización mensual. El programa deberá leer el monto de capital que se desea invertir, la tasa de interés anual y el número de meses que se desea tabular.

Para este programa, los datos de entrada son tres números: el monto del capital, la tasa de interés anual, y el número de meses a tabular. La salida del programa será una tabla de la forma:

```
Mes     Capital
---------------
  1     dddd.dd
  2     dddd.dd
...
```

Una primera aproximación a la solución a este problema puede ser el siguiente pseudocódigo:

```
lee capital, tasa y meses
genera tabla
```

La primera instrucción es una sentencia simple por lo que sólo necesitamos refinar es la segunda instrucción: `genera tabla`. Para generar esta tabla requerimos que la computadora repita el siguiente paso, tantas veces como meses se deseen en la tabla:

```
genera renglón
```

Para lograr que la sentencia compuesta se repita se utiliza la siguiente sentencia repetitiva:

```
mes = 1
mientras(mes <= meses) {
   genera renglón
   mes = mes + 1
}
```

La variable `mes` se utiliza como contador de las veces que se generan renglones de la tabla y debe inicializarse a uno ya que la primera vez que se ejecuta la sentencia corresponde al mes uno. El último paso de la sentencia compuesta anterior es necesario para que la computadora lleve la cuenta de cuantas veces ha ejecutado dicha sentencia. En este caso la variable mes se utiliza como un contador que se incrementa en uno cada vez que se ejecuta la sentencia.

Por lo tanto, una segunda aproximación a la solución estará dada por el siguiente pseudocódigo:

```
lee capital, tasa y meses

mes = 1
mientras(mes <= meses) {
   genera renglón
   mes = mes + 1
}
```

En el pseudocódigo anterior, sólo hay que refinar el paso `genera renglón`. Para generar cada renglón de la tabla se requieren los dos pasos siguientes:

```
calcula capital
escribe mes y capital
```

La variable `capital` en la que inicialmente se guardó el valor del capital a invertir, se usa para almacenar el valor del capital conforme se va acumulando mensualmente. Luego la tercera aproximación a la solución estará dada por el siguiente pseudocódigo:

```
lee capital, tasa y meses

mes = 1
mientras(mes <= meses) {
   calcula capital
   escribe mes y capital
   mes = mes + 1
}
```

En el pseudocódigo anterior, resta por refinar el paso de calcular el capital acumulado mensualmente. El capital acumulado en un mes se puede calcular a partir del capital acumulado en el mes anterior de la siguiente forma:

```
capital = capital + interés
```

esto es el capital de este mes es igual al capital del mes anterior más el interés ganado. El interés es el producto del capital por la tasa de interés mensual. Como el dato disponible es la tasa de interés anual, la dividimos entre 12 para obtener la tasa de interés mensual:

```
capital = capital + capital * tasa/12
```

o agrupando

```
capital  = capital * (1 + tasa/12)
```

Luego, nuestro pseudocódigo final es:

```
lee capital, tasa y meses

mes = 1
mientras(mes <= meses) {
   capital  = capital * (1 + tasa/12)
   escribe mes y capital
   mes = mes + 1
}
```

2. Crea un programa que calcule el área bajo la curva $y = x^2$ y que se encuentra entre las rectas $x = x_i$ y $x = x_f$. Aproxime el área bajo la curva como la suma de las áreas de n rectángulos inscritos bajo la curva, tal como se muestra en la figura 2-

1. El programa deberá pedir los valores de x_i y x_f, así como el número de rectángulos a usarse.

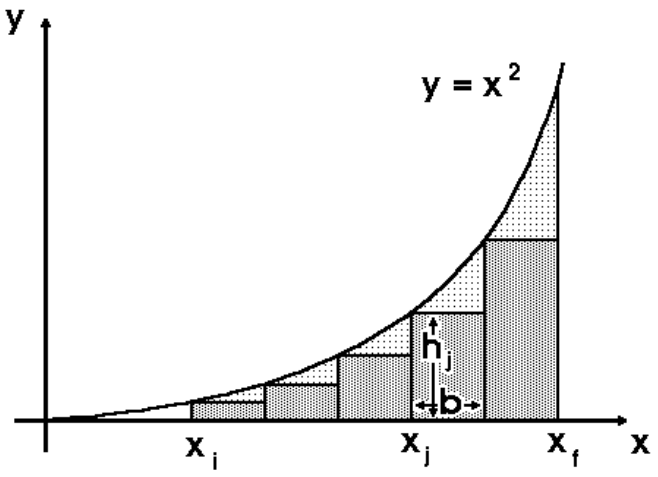

Figura 2-1

En este programa, los datos de entrada son tres números: los valores de las abscisas entre las que se calculará el área y el número de rectángulos que se usarán en éste cálculo. La salida del programa será un número, el área bajo la curva.

Una primera aproximación para el pseudocódigo de este programa es:

```
lee xi, xf, n
calcula área
escribe área
```

En el pseudocódigo anterior, el paso que se requiere desglosar más es él en que se calcula el área. Dado que el área bajo la curva se va a aproximar mediante el área de los n rectángulos inscritos bajo la curva, tenemos que:

$$\text{área} = \text{área}_1 + \text{área}_2 + \ldots + \text{área}_j + \ldots + \text{área}_n$$

Donde área_1, área_2, ..., área_j, ..., área_n son las áreas de los n rectángulos. Como el área de un rectángulo está dada por:

$$\text{área}_j = b \times h_j$$

Donde b y h_j son la base y la altura del rectángulo, respectivamente. El área de los n rectángulos es:

$$\text{área} = b \times h_1 + b \times h_2 + \ldots + b \times h_j + \ldots + b \times h_n$$

$$área = b \times (h_1 + h_2 + ... + h_j + ... + h_n)$$

Para obtener el área de los n rectángulos multiplicamos la base por la suma de las alturas de los n rectángulos. Luego el paso del cálculo del área bajo la curva puede desglosarse como:

```
calcula base
calcula sumaH
área = sumaH * base
```

La base de cada rectángulo puede calcularse mediante la expresión:

```
base = (xf - xi)/n
```

Para calcular la suma de las alturas requiere que se calcule cada una de las alturas de los rectángulos. La altura de cada rectángulo es el valor de la función para el valor de la abscisa izquierda de cada rectángulo: $y = x^2$. Por lo que la suma de las alturas puede calcularse mediante de la siguiente manera:

```
sumaH = 0
x = xi
mientras(x < xf) {
   y = x^2
   sumaH = sumaH + y
   x = x + base
}
```

El pseudocódigo final para este programa es:

```
lee xi, xf, n

base = (xf - xi)/n
sumaH = 0
x = xi
mientras(x < xf) {
   y = x^2
   sumaH = sumaH + y
   x = x + base
}
área = sumaH * base

escribe área
```

Ejercicios sobre la Sentencia Repetitiva

1. Escribir el pseudocódigo para un programa que lea las edades de un grupo de alumnos y encuentre la edad promedio.

2. Escribir el pseudocódigo para un programa que determine el número de meses necesario para que una inversión, colocada a una tasa de interés anual dada y con recapitalización mensual, se duplique.

Programación Estructurada

Las tres estructuras de control vistas, la **sentencia compuesta**:

```
sentencia1
 [sentencia2]...
```

la **sentencia condicional**:

```
si(expresión) sentencia1
[otro sentencia2]
```

y la **sentencia repetitiva**:

```
mientras(expresión)
   sentencia
```

todas tienen un sólo punto de entrada o inicio y un sólo punto de salida o final. Su punto de entrada es la parte superior y su punto de salida está en la parte inferior. Lo anterior nos permitirá crear programas más complejos combinando las estructuras de control de diferentes maneras, uniendo las salidas de unas con las entradas de otras. Para este fin existe una **regla fundamental de composición** que establece lo siguiente:

> Las estructuras de control que se pueden formar combinando de manera válida las sentencias secuenciales, condicionales y repetitivas también serán válidas.

Lo anterior da origen al concepto de **programación estructurada**. Un programa estará bien construido si está formado por estructuras de control válidas, de acuerdo con la regla precedente.

Por ejemplo, podemos construir una sentencia secuencial a partir de una sentencia secuencial y una sentencia condicional:

```
sentencia1
sentencia2

si(expresión) {
   sentencia3
   sentencia4
}
otro {
   sentencia5
   sentencia6
}
```

Note que esta nueva sentencia sigue teniendo una sola entrada y una sola salida.

Un segundo ejemplo es una sentencia condicional donde su primera sentencia compuesta está formada de otra sentencia condicional seguida de una sentencia iterativa y su segunda sentencia también contiene una sentencia condicional:

```
si(expresión1) {
   si(expresión2) {
      sentencia1
      sentencia2
   }
   otro {
      sentencia3
      sentencia4
   }

   mientras(expresión3) {
       sentencia5
       sentencia6
   }
}
otro
   si(expresión4) {
      sentencia7
      sentencia8
   }
   otro {
      sentencia9
      sentencia10
   }
```

Ejemplo Sobre Programación Estructurada

Modifique el pseudocódigo del ejemplo 2 sobre la sentencia condicional para que la calculadora sea capaz de realizar operaciones consecutivas, mostrando los resultados parciales. Por ejemplo si tecleamos:

```
4
+
3
```

la computadora nos mostrará el resultado de la suma:

```
4
+
3
7
```

y estará esperando que tecleemos el operador y operando para la siguiente operación. Por ejemplo supongamos que al resultado anterior le deseamos restar 5. Entonces teclearemos:

```
4
+
3
7
-
5
```

La computadora desplegará el resultado de la resta:

```
4
+
3
7
-
5
2
```

y volverá a esperar que tecleemos el operador y operando para la siguiente operación. Si deseamos que el programa termine presionaremos la tecla '='.

En este programa la entrada estará formada por números que representan los operandos y caracteres que representan los operadores. La salida consistirá de números que representan los resultados parciales y mensajes de error si la operación es inválida.

Una primera aproximación al programa está dada por el siguiente pseudocódigo:

```
lee resultado
lee operador
mientras(operador != '=') {
   lee operando
   calcula resultado
   escribe resultado
   lee operador
}
```

En el primer paso del programa, estamos leyendo el primer número y lo estamos almacenando en una variable llamada resultado. En el segundo paso leemos el carácter que representa la operación a realizar. En la expresión de la sentencia repetitiva:

```
operador != '='
```

el símbolo **!=** significa "diferente de", por lo que el ciclo se repetirá mientras el operador sea diferente del carácter '='.

En el primer paso de la sentencia compuesta de la sentencia repetitiva leemos el otro número con el que se va hacer la operación y lo almacenamos en la variable operando.

El paso en que se calcula el resultado es similar al del ejemplo 2 sobre la sentencia condicional, por lo que el pseudocódigo del programa queda como:

```
lee resultado
lee operador

mientras(operador != '=') {
   lee operando

   si(operador == '+') resultado = resultado + operando
   otro si(operador == '-') resultado = resultado - operando
   otro si(operador == '*') resultado = resultado * operando
```

```
    otro si(operador == '/')
        si(operando != 0.0) resultado = resultado / operando
        otro escribe "Error: División entre cero"
    otro escribe "Error: Operador desconocido"

    escribe resultado

    lee operador
}
```

Problemas

1. Escribir el pseudocódigo para un programa que lea un número que represente una distancia en metros y escriba su equivalente en pies y en pulgadas.

2. Escribir el pseudocódigo para un programa que lea un número que represente el radio de una esfera y escriba su área y volumen.

3. Escriba el pseudocódigo para un programa que indique a una cajera de banco el número y denominación de los billetes que necesita darle a un cliente al hacer un retiro. La cajera deberá darle al cliente billetes de la más alta denominación posible, esto es, el menor número de billetes. Suponga que los retiros deben de ser en cantidades múltiples de 10 pesos y que hay billetes de $10, $20, $50 y $100 pesos.

4. Un año bisiesto es aquel año que es divisible entre 4 pero no es divisible entre 100 a menos que sea divisible entre 400 en cuyo caso si es bisiesto. Crea el pseudocódigo de un programa que determine si un año dado es un año bisiesto o no.

5. La capacitancia de un capacitor de placas paralelas con vacío por dieléctrico está dada por:

$$C = \varepsilon_0 \frac{A}{d}$$

   Donde $\varepsilon_0 = 8.85 \times 10^{-12}$ F/m es la permitividad del vacío, A es el área de una de las placas del capacitor y d es la distancia entre las placas.

   Crea el pseudocódigo de un programa que calcule la capacitancia de un capacitor de placas paralelas. Las placas paralelas pueden tener forma rectangular o circular. El programa pedirá el tipo de placas del capacitor, ‘R’ o ‘C’ y dependiendo del tipo las dimensiones a y b de los lados del rectángulo o el radio r. Adicionalmente el programa pedirá la distancia d, entre placas.

6. Escribir el pseudocódigo para un programa que genere una tabla de los cuadrados y los cubos de los números enteros del 1 al 10. La tabla resultante debe tener la siguiente forma:

```
Número    Cuadrado     Cubo
---------------------------------
  1          1          1
  2          4          8
  etc.
```

7. Escribir el pseudocódigo para un programa que tabule la ecuación:

$$y = x^3 - 2x + 3$$

   El programa deberá pedir los límites inferior, superior y el valor del incremento de x.

8. Crea el pseudocódigo para un programa que calcule el promedio de cada alumno de un grupo, para lo cual se leerán su matrícula y cuatro calificaciones. Indicar fin de datos con matrícula = 0. El programa deberá imprimir la matrícula, las cuatro calificaciones y el promedio de cada alumno. Al final deberá imprimir la calificación promedio global del grupo.