Elementos de Programación

UNIDAD 2. ESTRUCTURA SECUENCIAL

INDICE

UNIDAD 2 - Estructura Secuencial

OBJETIVOS: *Construir algoritmos utilizando la programación estructurada. Resolver sencillos ejemplos con "estructura secuencial" representándolos gráficamente mediante diagrama de flujo.*

1. Estructura Secuencial

La primera estructura básica de la programación estructurada es la secuencial. Consiste en una sucesión de instrucciones que se ejecutan en el mismo orden en que fueron escritas, una a continuación de la otra.
Un algoritmo secuencial se expresa como una sucesión de instrucciones que se ejecutan TODAS, una a continuación de la otra, teniendo un único punto de inicio y un único punto de fin. Esas instrucciones pueden ser de distintos tipos:

- Operaciones aritméticas
- Operaciones de asignación
- Ingreso de datos
- Salida / Mostrar información

2. Almacenamiento de los datos

Antes de poder escribir las instrucciones que forman parte de los algoritmos, debe saber cómo se guardan los datos internamente en la computadora, ya que las instrucciones serán operaciones que se realizarán sobre dichos datos.

Cada fragmento de información almacenado en la memoria de la computadora es codificado como una combinación de ceros y unos. Estos ceros y unos se llaman bits (dígitos binarios). Un dispositivo electrónico representa cada BIT, el cual estará de alguna forma: "apagado" (cero) o "encendido" (uno).

A un grupo de 8 bit, se lo denomina byte. Normalmente, un carácter (una letra, un dígito, un carácter especial) ocupara un byte de memoria.

2.1 Variable:

Una variable es un espacio de memoria reservado para almacenar un valor que corresponde a un tipo de dato soportado por el lenguaje de programación. Una variable, es representada y usada a través de una etiqueta (un nombre) que le asigna un programador, o que ya viene predefinida.

Por ejemplo, se define una variable con el nombre NUM, en la cual se almacena el número 8, NUM es la etiqueta que le asigna el programador y ocho es su contenido. Lo que se carga en NUM puede ser cualquier número, 1, 2, 3, 0, 12, 33, etc., (variable). Una variable, por lo general, como su nombre lo indica, puede variar su valor durante la ejecución del programa.

2.2 Tipos de Datos:

Los tipos de datos indican que es lo que se puede almacenar en una variable. Por ejemplo, un número entero, un carácter, un número con decimales, etc.

Cada tipo de dato ocupa en la memoria de la computadora una cantidad distinta de bytes. Por lo tanto, para optimizar un programa, por ejemplo, si se necesita trabajar con números pequeños, se puede utilizar una variable con un tipo de dato que permita almacenar números pero hasta cierta cantidad haciendo que esa variable ocupe menos espacio en memoria.

Dependiendo del lenguaje de programación usado, también es posible cambiar el tipo de dato que almacena una variable a lo largo del programa, pero en el lenguaje C, que es el que se verá en esta materia, el tipo de dato no puede modificarse, por lo que debe definirse la variable correctamente según lo que se necesite almacenar en ella.

Los tipos de dato que se utilizarán para los algoritmos son los siguientes:

- **Caracter**: para almacenar un único caracter, una letra, un símbolo, un dígito numérico.
- **Entero**: para almacenar un número entero (sin decimales).
- **Real**: para almacenar un número con decimales.

2.3 Identificadores:

Los nombres de variables o identificadores deben ser nemotécnicos, es decir, que con solo leer el nombre de la variable se pueda entender o determinar con facilidad lo que ella significa o contiene. En el ejemplo anterior la variable con el nombre NUM almacena un número por lo que se ve a simple vista, pero si a esta variable se le hubiese dado el nombre X o DS, estos nombres pueden significar muchas cosas o, tal vez, no significar absolutamente nada haciendo que lectura del programa sea más compleja. Los identificadores tienen ciertas reglas de escritura que van a depender del lenguaje de programación que se esté utilizando. Como regla general, un identificador tiene que cumplir con las siguientes reglas:

- No puede comenzar con números.
- No puede tener espacios.
- No puede tener acentos ni letra ñ.
- No puede tener símbolos (excepto el guión bajo que es el único permitido).

A continuación se muestra la tabla 1 con ejemplos de identificadores:

Tabla 1: ejemplo de identificadores

Identificador	¿es correcto?
Num	CORRECTO
Numero_1	CORRECTO
_Numero1	CORRECTO
Num1	CORRECTO
Número	INCORRECTO
1Numero	INCORRECTO
Num 1	INCORRECTO
año	INCORRECTO

Algunos lenguajes son sensibles a los cambios entre mayúsculas y minúsculas (case-sensitive), y otros no. En aquellos que son sensibles, por ejemplo num1 y Num1, hacen referencia a dos identificadores diferentes por lo que debe respetarse siempre el mismo nombre cada vez que se lo utiliza.

2.4 Constante:

Constante son todos aquellos valores que no cambian en el transcurso de un algoritmo y son introducidos en el momento de utilizarse.

Por ejemplo, el número 10 es una constante del tipo entera que puede asignarse a una variable de dicho tipo. En cambio, si se quiere representar una constante del tipo caracter se expresa entre comillas simples, por ejemplo, 'A' , '1', 'a' , '-' son constantes de tipo caracter.

3. Representación gráfica de un algoritmo - Diagrama

Para comenzar a diseñar los algoritmos, y ayudar a encontrar una solución al problema planteado, se va a utilizar una representación gráfica de las instrucciones que se le darán a la computadora, de esta forma se podrá visualizar más fácilmente el algoritmo y una vez terminado y probado se puede codificar en el lenguaje de programación elegido.

Todo diagrama tendrá un símbolo de inicio, una serie de operaciones y un símbolo que indica el fin del mismo. Recuerde que inicialmente realizará algoritmos en forma secuencial, por lo tanto, los símbolos que utilizará son los que están en la Figura 1.

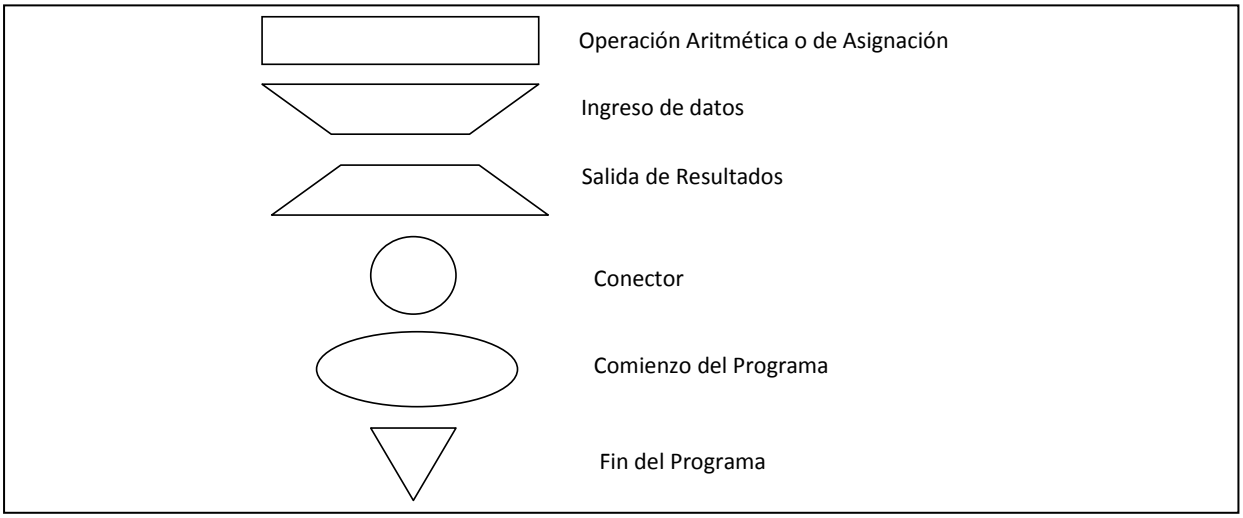


Figura 1. Símbolos para la construcción de diagramas que representan algoritmos secuenciales.

3.1 Ingreso de datos

Para resolver un problema, se necesitan datos, esos datos serán procesados para alcanzar el resultado esperado.

Los algoritmos, en general, se diseñan para que puedan funcionar con distintos datos según la necesidad del usuario que esté utilizando el programa. Por ejemplo, poca utilidad tendrá un programa que calcule una suma de números pre-establecidos: si se suma 5 + 2, siempre el resultado será 7, por lo tanto, hacer un programa con esos números fijos (constantes) no tiene mucha utilidad. En cambio si esos dos números a sumar pueden ingresarse y variar cada vez que se ejecute el programa, se podrá utilizar para sumar 5 + 2, 9 + 4, 25458 + 24577, o para cualquier par de números que se desee.

Analice el problema a resolver: Se necesita realizar un programa que permita sumar dos números cualquiera. Una forma de darle la información a la computadora es ingresar ese número por teclado, ¿pero una vez ingresado donde queda ese número? Bien, se necesita entonces un lugar para guardar cada uno de los números que se ingresan en el programa, y como vimos anteriormente, los datos se guardan en la memoria de la computadora en espacios reservados que se llaman variables.

El programa va a necesitar entonces dos variables para los datos de entrada, una variable para guardar el primer número a sumar, y la otra para guardar el segundo número a sumar:

- En la variable que se llama Num1 se va a guardar el primer número
- En la variable que se llama Num2 se va a guardar el segundo número

En el algoritmo cuando se quieran ingresar uno o más datos se utilizará el símbolo de ingreso, y dentro del símbolo se pondrá el nombre de la o las variables donde se guardará la información a ingresar. Si se ingresa más de un dato, las variables se pondrán separadas por coma. La figura 2 muestra la presentación gráfica para permitir el ingreso de datos desde teclado y que se almacenen los datos ingresados en las variables Num1 y Num2.

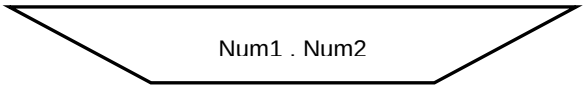


Figura 2. Simbología para el ingreso de dos datos que se almacenarán en las variables Num1 y Num2

Num1 y Num2 son dos variables distintas. Num1 va a contener el primer dato que el usuario ingrese por teclado, y Num2 el segundo. Antes de ingresar los datos en dichas variables no se sabe que hay, por lo tanto, no se pueden utilizar si antes no se leyó un dato o se les asignó un valor (proceso que se verá a continuación).

3.2 Operaciones

Sobre los datos ingresados se realizan operaciones para obtener el resultado deseado. Se pueden realizar operaciones matemáticas y de asignación.

3.2.1 Operación de Asignación

La asignación consiste en guardar un dato en una variable. El dato puede ser:

- Otra variable
- Una constante
- El resultado de una operación matemática

La asignación se realiza de derecha a izquierda, es decir, que a la izquierda se escribe el identificador de la variable en la cual desea guardar el dato, luego se escribe el símbolo = (igual) que indica la operación de asignación, y a la derecha se coloca la constante, variable u operación matemática. Gráficamente la operación de asignación se representa con un rectángulo. En la Figura 3 pueden verse distintos ejemplos de asignación.

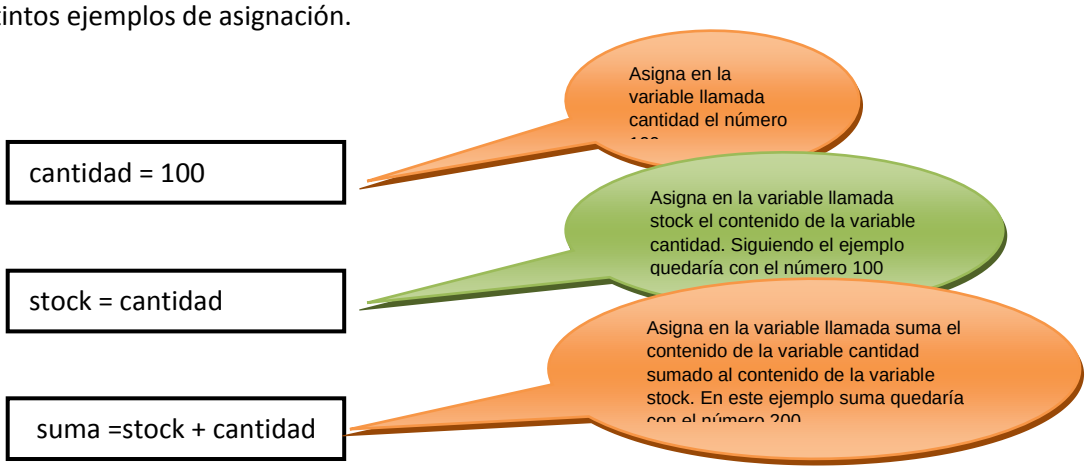


Figura 3. Ejemplos de asignación

3.2.2 Operaciones matemáticas

Sobre los datos se pueden realizar distintas operaciones. En la tabla 2 se detallan las operaciones posibles y el símbolo que se utiliza para expresarlas. A estos símbolos se los denomina operadores matemáticos.

Tabla 2. Operadores Matemáticos

Operación	Operador matemático
Suma	+
Resta	-
Multiplicación	*
División	/
Resto de la división entera	%

Las operaciones se pueden realizar tanto sobre variables (tomando el contenido que tienen almacenado), como sobre constantes. En la figura 4 pueden verse distintos ejemplos de operaciones que se realizan una a continuación de la otra.

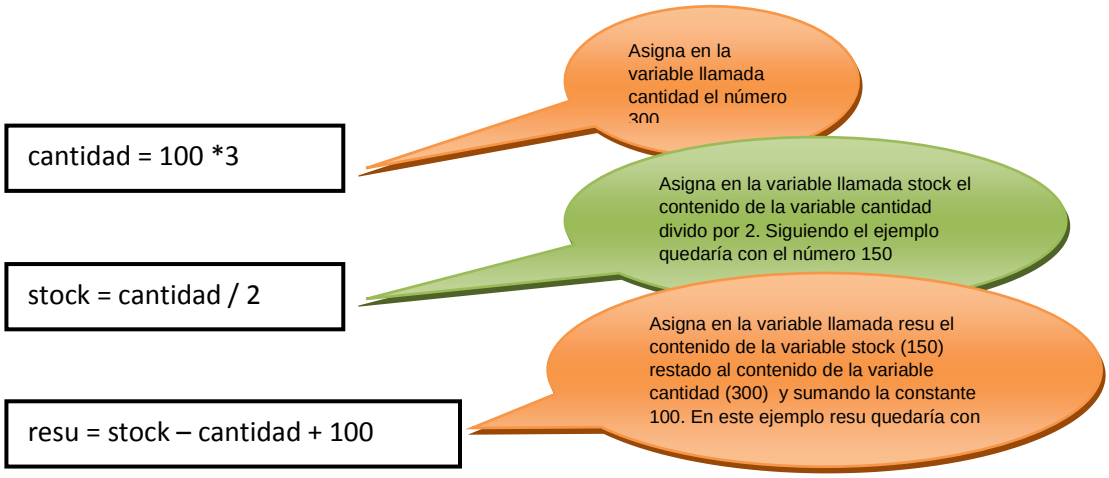


Figura 4. Ejemplos de operaciones matemáticas

Una mención especial requiere la operación de división ya que se comporta diferente según sea el tipo de dato de las variables o constantes sobre las que se está operando. Por ejemplo, al dividir el numero 15 con el numero 2 (15 / 2) como ambas son constantes de tipo entero la operación dará por resultado la de división entera. Es decir, que el resultado de 15 / 2 será 7 y no 7.5 como sería lo esperado. Esta característica hay que tenerla en cuenta al realizar las operaciones ya que puede llevar a resultados erróneos pero brinda la posibilidad de realizar muchas otras tareas de forma sencilla, como por ejemplo, al descomponer un número. Siguiendo el ejemplo anterior si se quiere obtener 7.5 como resultado al menos una de las constantes debe ser del tipo real, es decir, un número con decimales. Como separador de decimales se usa el punto, por lo tanto, el resultado de 15. / 2 será 7.5. El mismo resultado puede obtenerse al realizar la operación 15 / 2. .

Los operadores tienen precedencia al igual que en matemática, por ejemplo, el operador suma (+) separa términos y se ejecuta luego de realizar las operaciones de mayor precedencia como la multiplicación o la división. Muchas veces es necesario cambiar dicha precedencia y, por lo tanto, debemos agrupar las operaciones. Para ello se utilizan paréntesis. En una expresión pueden utilizarse todos los niveles de paréntesis que sean necesarios (NO corchetes). Por ejemplo, en la siguiente instrucción:

```
suma = 3 + 2 * 5
```

La variable `suma` quedará con el valor 13 ya que primero realiza la multiplicación y luego la suma. En cambio, si la misma instrucción se escribiera de la siguiente manera:

```
suma = (3 + 2) * 5
```

La variable `suma` quedará con el valor 25 ya que se utilizaron los paréntesis para resolver la suma primero que la multiplicación.

3.2.3 Salida de Resultados / Mensajes

Una vez realizado los cálculos sobre los datos el resultado debe mostrarse al usuario. Para ello, se utiliza la simbología que se muestra en la figura 5, donde pueden verse distintas formas de mostrar un resultado, donde se puede combinar datos con mensajes literales. Por defecto, la salida se realizará por pantalla, es decir, que se especifica lo que el usuario va a visualizar en la pantalla. Los mensajes se escriben entre comillas, haciendo que todo lo que se ponga entre las comillas se vea tal cual en la pantalla. Ahora si se desea mostrar el resultado de una operación que está almacenado en una variable, el identificador de la variable NO se pone entre comillas, de esa forma lo que se verá en pantalla es el contenido que tiene la variable en ese momento.

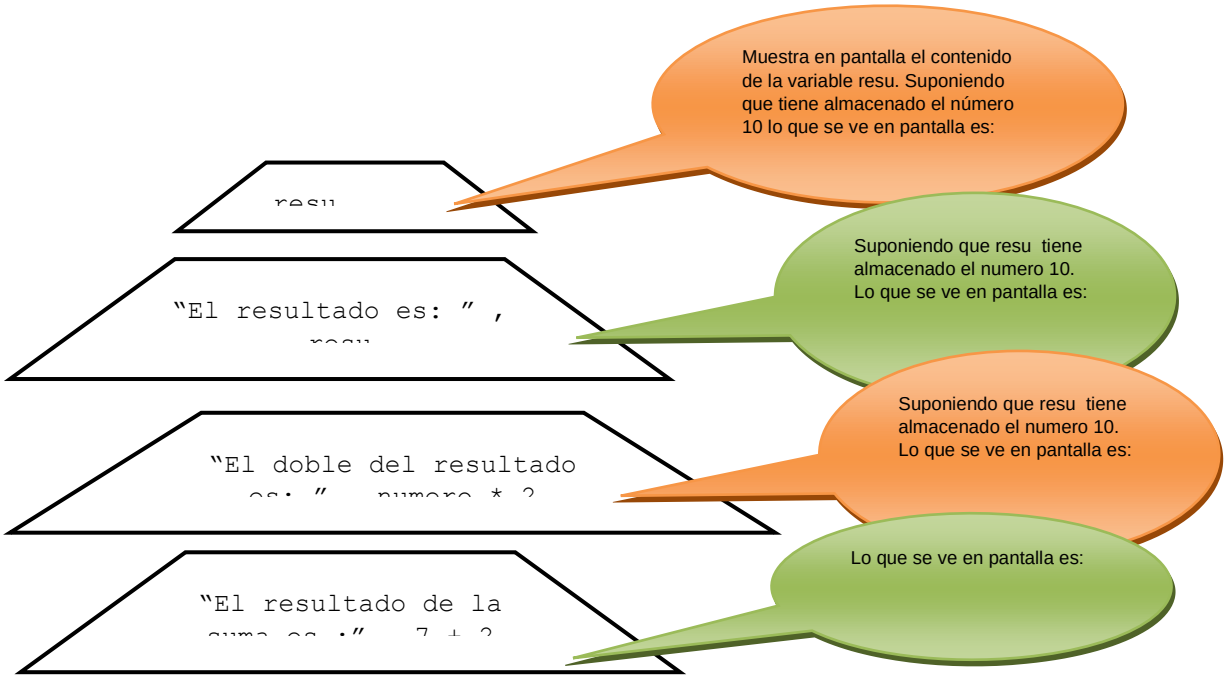


Figura 5. Ejemplos de salida por pantalla

3.2.4 Símbolos de indicación

Además de la entrada, salida y proceso hay tres símbolos adicionales que se utilizan para dar una mayor legibilidad al diagrama. Todo diagrama comenzará con un óvalo que indica el comienzo del algoritmo, dentro de este óvalo opcionalmente se puede escribir un nombre que represente el algoritmo que se está desarrollando. Al finalizar y para cerrar el diagrama se utilizará un triángulo invertido donde opcionalmente dentro se puede poner la letra F que indica fin. Por último, si se está desarrollando un diagrama largo que no entra en una hoja y se desea continuar en la hoja siguiente se utiliza un círculo como conector de la siguiente forma: al final de la hoja en lugar de la instrucción siguiente se pone el círculo con un número dentro, por ejemplo el número 1, ya que es el primer conector del algoritmo, luego en la hoja siguiente se repite en la parte superior el círculo con el mismo número para indicar que allí continua el diagrama. La Figura 6 muestra un ejemplo de la utilización de los símbolos de indicación.

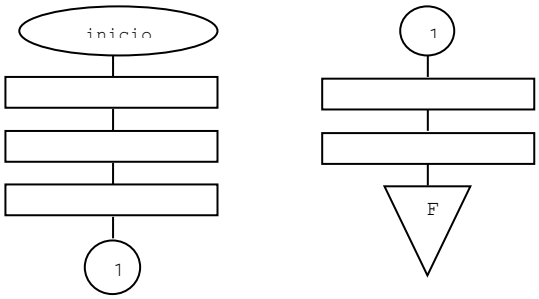
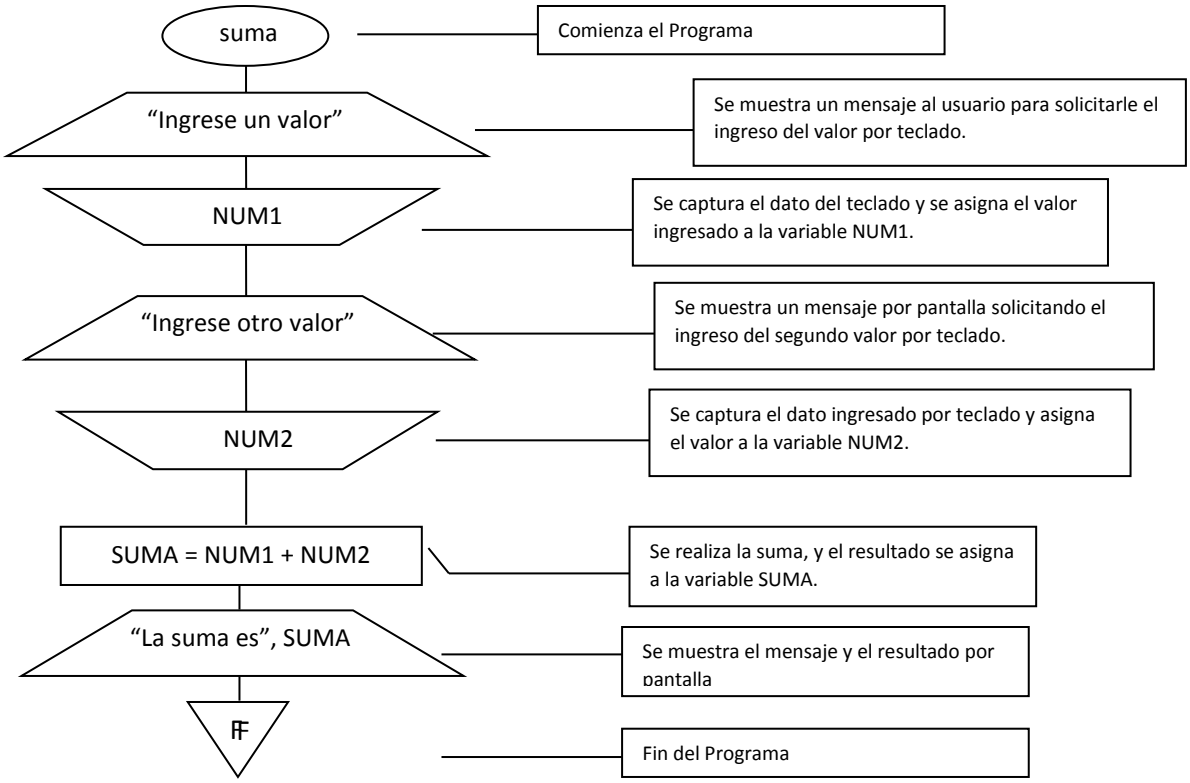


Figura 6. Ejemplos de utilización de símbolos de indicación

4. Algunos ejemplos

Ejemplo 1

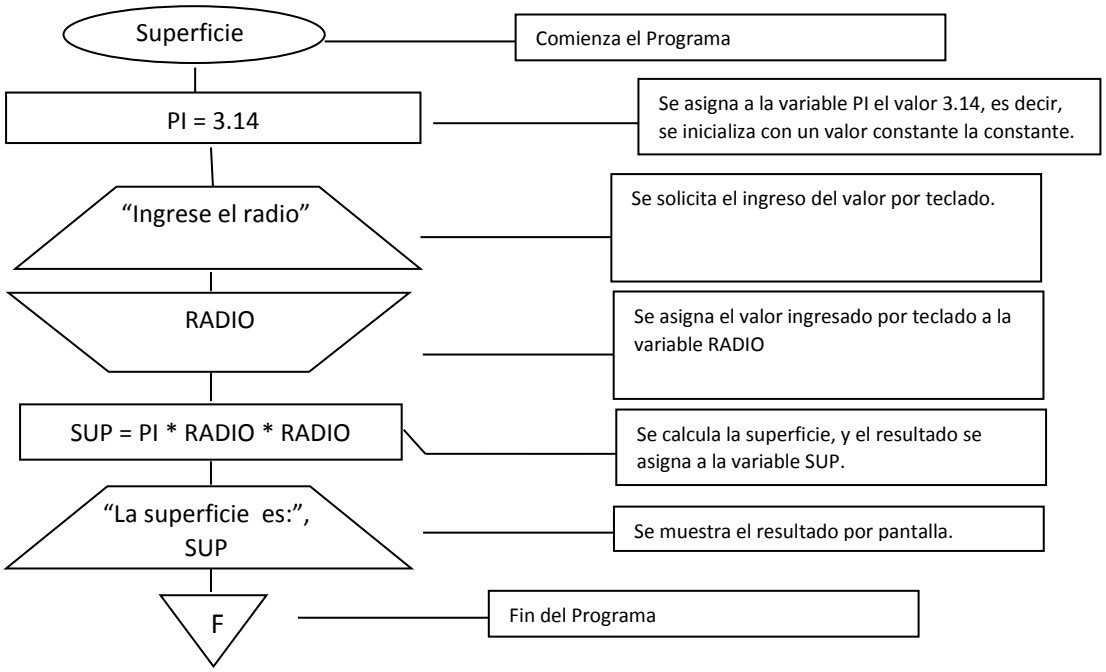
Realizar un programa que permita ingresar dos números enteros por teclado, realice la suma, y muestre por pantalla el resultado.



En este ejemplo se incorporaron al diagrama todos los mensajes que se muestran al usuario para indicar lo que debe hacer, en ejercicios siguientes, más complejos, los mensajes al usuario pueden ser omitidos para facilitar el diagrama, pero luego sí, al codificar en el lenguaje de programación, es muy importante siempre poner mensajes aclaratorios indicándole al usuario lo que debe hacer y cómo se comporta el programa.

Ejemplo 2

Realizar un programa que permita ingresar el radio de un círculo, calcule la superficie, y muestre por pantalla el resultado.



Ejemplo 3

Realizar un diagrama que permita ingresar un número de dos cifras, y muestre por pantalla las unidades y las decenas.