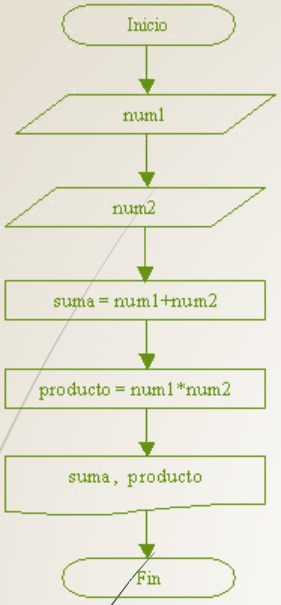
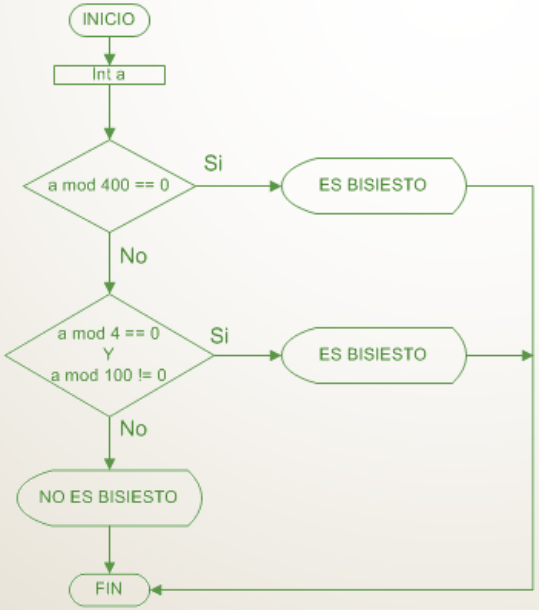


Programación Estructurada

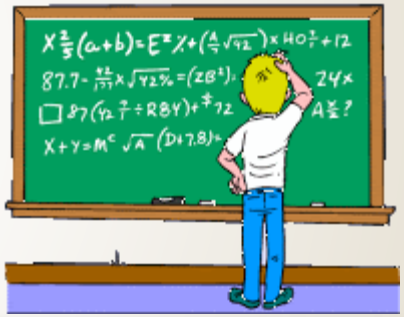
Modularidad: Funciones



Facultad de Ingeniería
Universidad Nacional de Jujuy

Complejidad de problemas (1)

- Los problemas sencillos pueden resolverse mediante pequeños programas que ejecutan un conjunto simple de instrucciones.
- Los problemas complejos deben ser tratados por programas que combinen varias instrucciones y éstas se resuelvan las distintas partes que hacen al problema.



Complejidad de problemas (2)

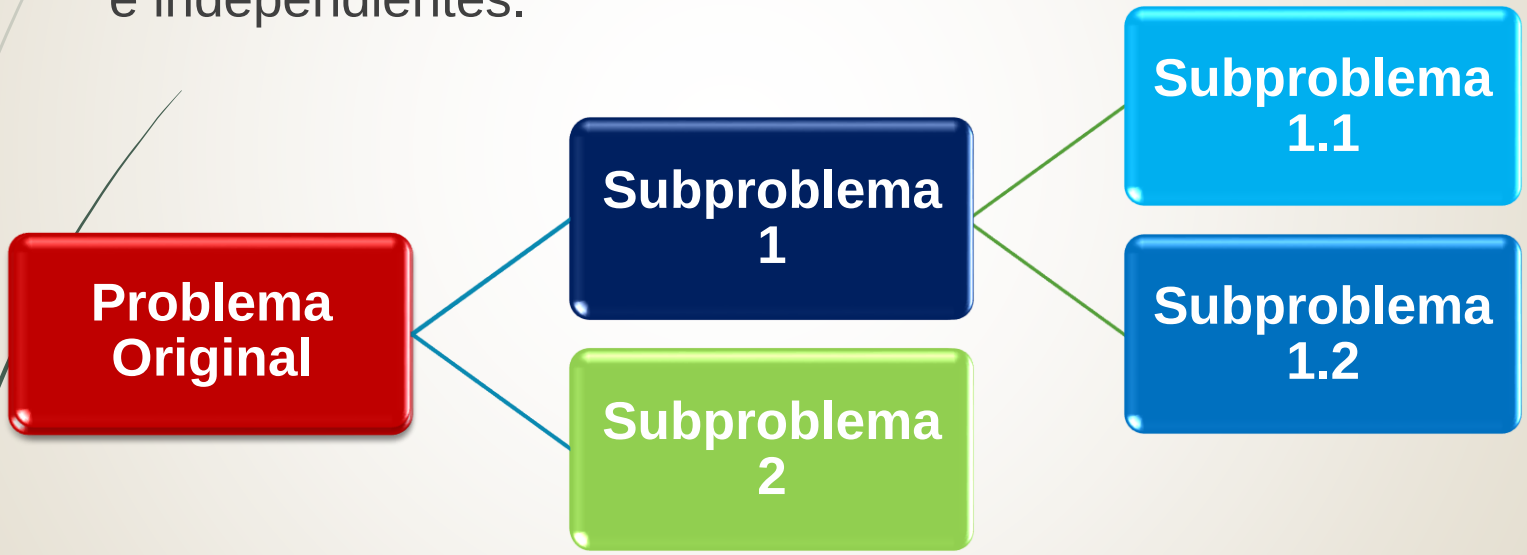
- Un programa compuesto por cientos o miles de instrucciones puede ser difícil de leer, seguir (entender) y probar.
- Para reducir la complejidad de un programa, éste puede dividirse en unidades de trabajo.
- Estas unidades se enfocan en tareas simples que cuyo funcionamiento combinado permite resolver el problema complejo.

Programación Modular (1)

- La programación modular
  - es un método de diseño flexible
  - permite dividir un programa en unidades de trabajo (subprogramas)
  - descomposición de problemas, abstracción y módulos
- Subprogramas o módulos
  - pueden analizarse, codificarse y probarse por separado
  - el módulo principal controla el flujo de acciones
  - se clasifican en Procedimientos y Funciones

Programación Modular (2)

- Descomposición de problemas
  - Un problema complejo puede dividirse en problemas sencillos e independientes.



Programación Modular (3)

- Abstracción
  - Permite representar los objetos relevantes del problema.

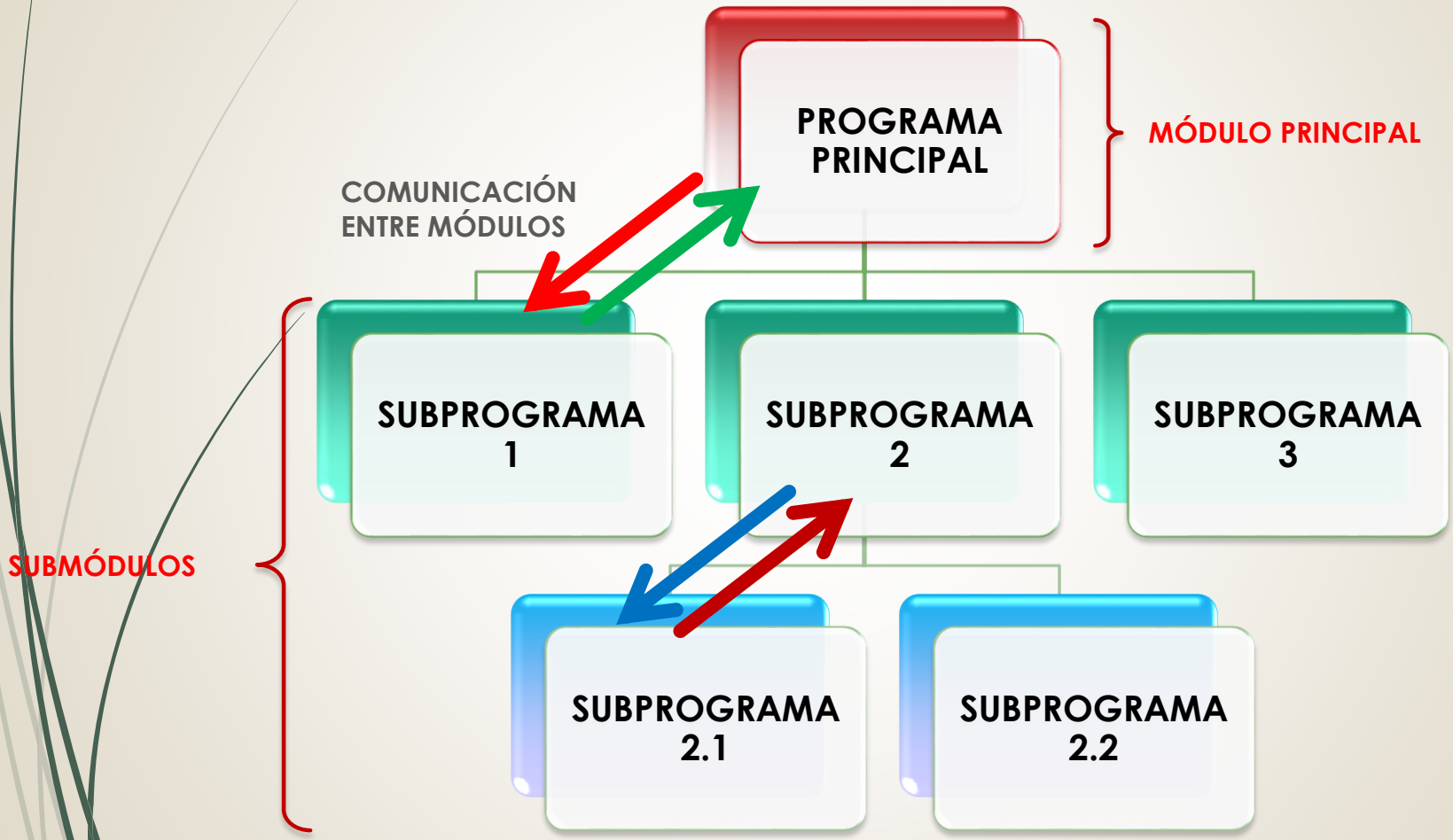
Programación Modular (4)

- Módulos
  - Un programa puede estar formado por partes independientes que resuelven subproblemas específicos.

Programación Modular (5)

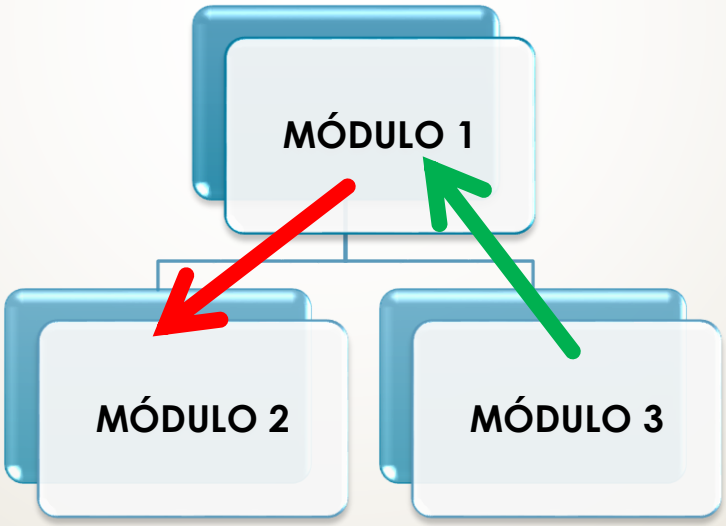
- Entradas: se conoce el conjunto de datos con los que trabajará el módulo.
- Propósito: se conoce el objetivo del módulo (qué hace).
- Salidas: se conoce el resultado que generará el módulo.

Programación Modular (6)



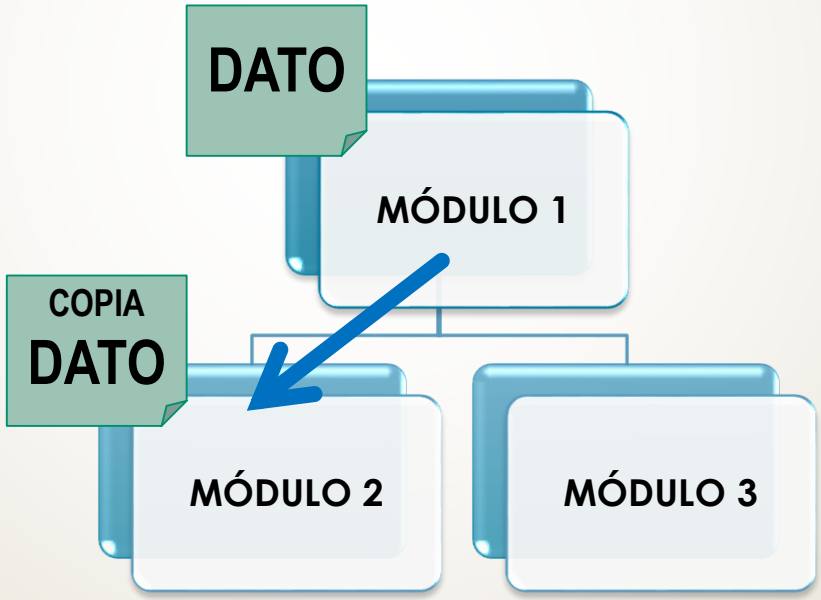
Comunicación entre módulos

- Los datos que usan los módulos de un programa se comunican a éstos cuando son invocados. Esta comunicación se denomina *Pasaje de Parámetros*.



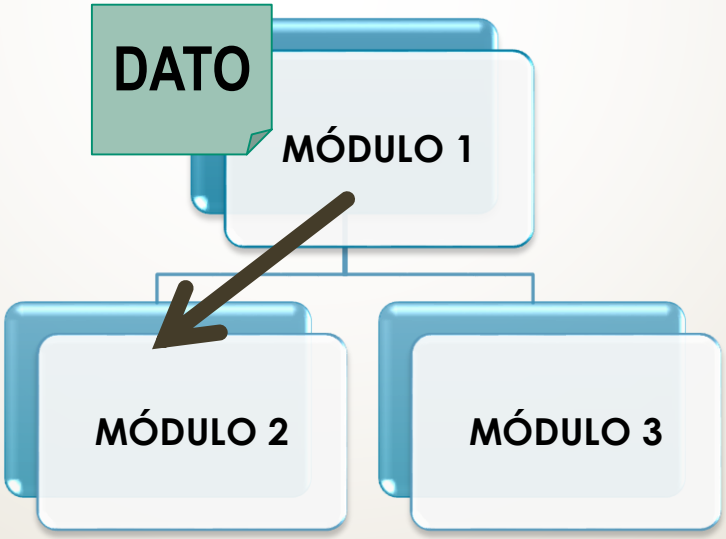
Pasaje de Parámetros (1)

- Por Valor: el módulo trabaja con copias de los datos originales. Estos parámetros se conocen como de entrada (E).



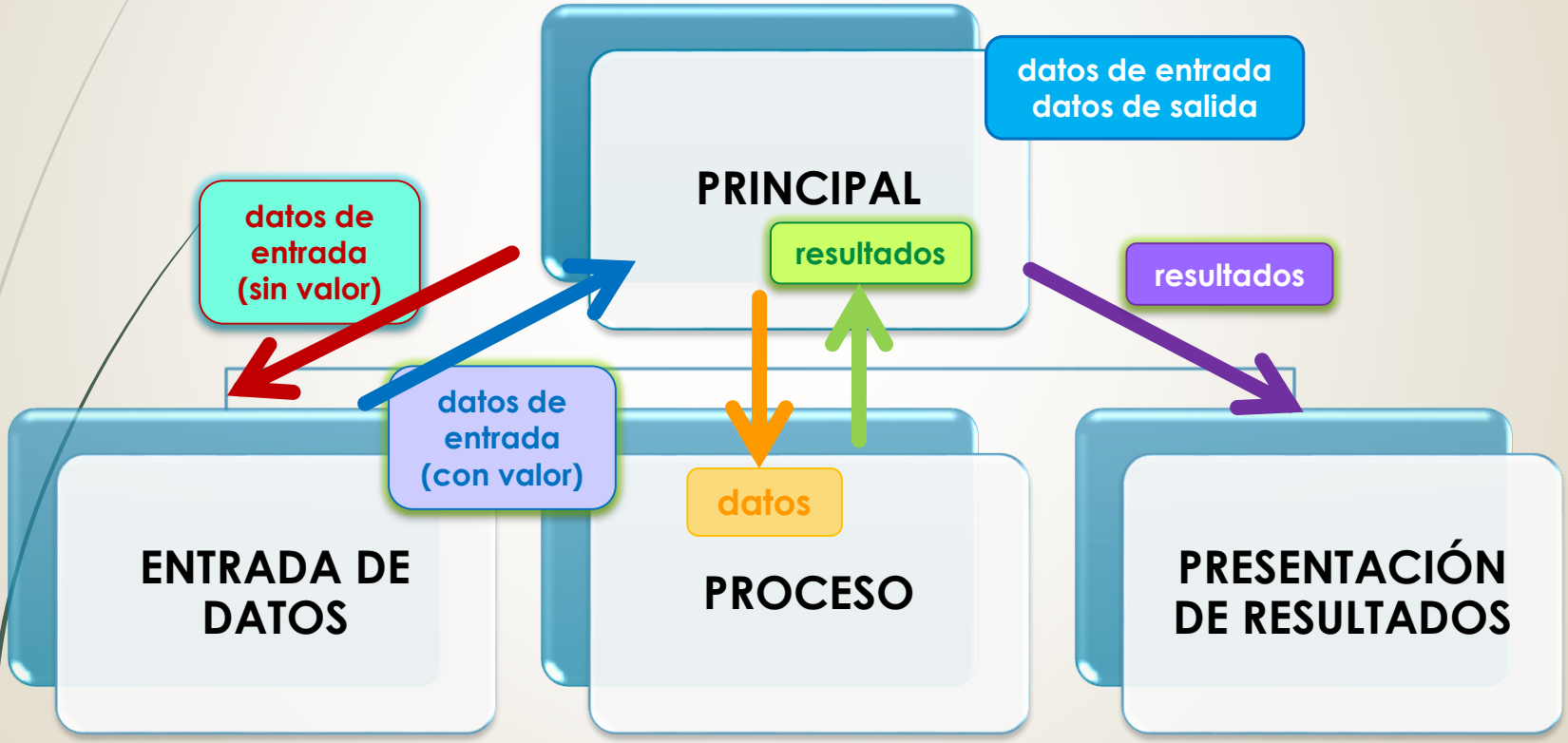
Pasaje de Parámetros (2)

- Por Referencia: el módulo trabaja con los datos originales y cualquier modificación altera los datos del programa que invocó al módulo. Estos parámetros se conocen como de entrada y salida (E/S).



Pasaje de Parámetros (3)

- Comunicación entre módulos de un programa



Funciones (1)

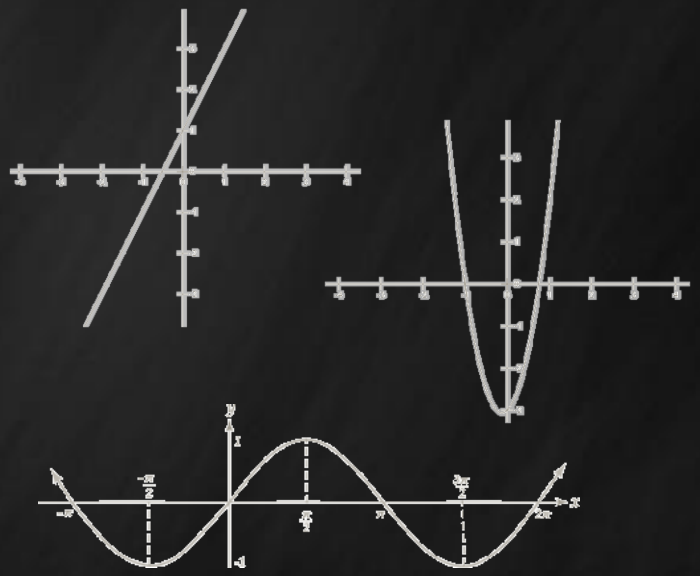
Las funciones matemáticas se definen como la expresión matemática de la relación existente entre dos variables o magnitudes.

Por ejemplo:

$F(x) = 2x + 1$

$F(x) = 4x^2 + x - 3$

$F(x) = \text{sen } x$



Funciones (2)

Parámetros Formales: son valores genéricos (variables) con los que se define la función.

Por ejemplo: F(x) = 2 x + 1

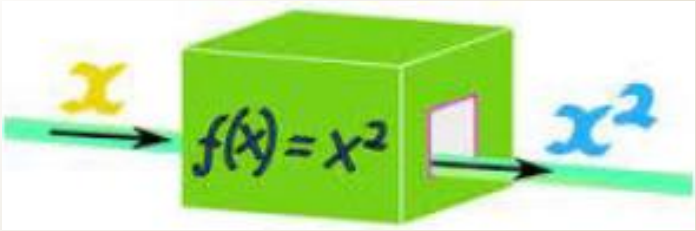
Parámetros Actuales: son valores específicos que utilizará la función para calcular un resultado.

Por ejemplo: F(3) = 2 . 3 + 1

F(3) = 7

Funciones (3)

- Una función es un módulo o subprograma que toma una lista de valores llamados argumentos o parámetros y devuelve un único valor.
- Las funciones se definen de un tipo de dato simple (entero, real, carácter, lógico).
- Las funciones pueden ser internas o definidas por el usuario.



Declaración de funciones (1)

```
FUNCIÓN Nombre_función (Parámetros formales): Tipo_de_Función
VARIABLES
   Variables_de_la_Función
INICIO
   ACCIONES
   Nombre_función←resultado_de_la_función
FIN
```

- *Nombre_función:* especifica el nombre de la función.
- *Parámetros Formales:* son los valores que recibe la función y que se usarán en el cálculo.
- *Tipo_de_Función:* la función puede ser entera, real, carácter, lógica.
- *Variables_de_la_Función:* son las variables de la función, están definidas para la función y desaparecen cuando ésta finaliza su ejecución.
- *ACCIONES:* sentencias secuenciales, selectivas o repetitivas que implementan la operación.
- *Nombre_función←resultado_de_la_función:* el resultado del cálculo de la función se asigna al nombre de la función y se retorna al programa que la invocó.

Declaración de funciones (2)

```
tipo_función nombre_función (parámetros formales)
{ tipo_dato nombre_variables; //variables de la función
  ACCIONES;
  return resultado_de_la_función;
}
```

- **tipo_función**: indica el tipo de resultado que devolverá la función.
- **parámetros formales**: indica los valores (y sus tipos) que utilizará la función.
- **variables de la función**: son las variables creadas sólo para la función (locales).
- **return resultado**: asigna el resultado final a la función.

Invocación de funciones

- Un función puede invocarse:

```
variable←nombre_función(parámetros_actuales)
```

```
variable=nombre_funcion(parámetros_actuales);
```

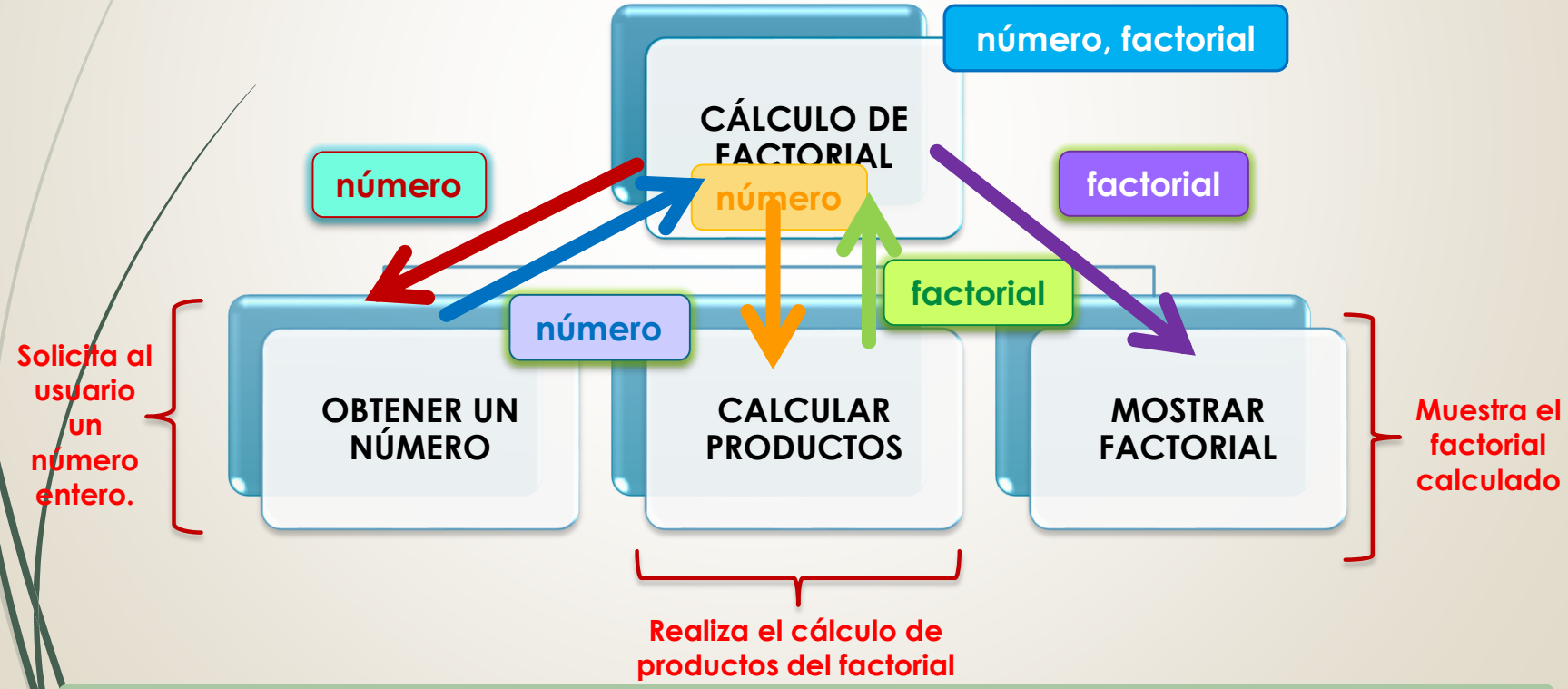
```
ESCRIBIR "Resultado:", nombre_función(parámetros_actuales)
```

```
cout << "Resultado:" << nombre_función(parámetros_actuales);
```

- Al invocar una función:
  1. A cada parámetro formal se le asigna el valor de su correspondiente parámetro actual.
  2. Se ejecuta el cuerpo de acciones de la función.
  3. Se asigna el resultado a la función y se retorna al punto de llamada.

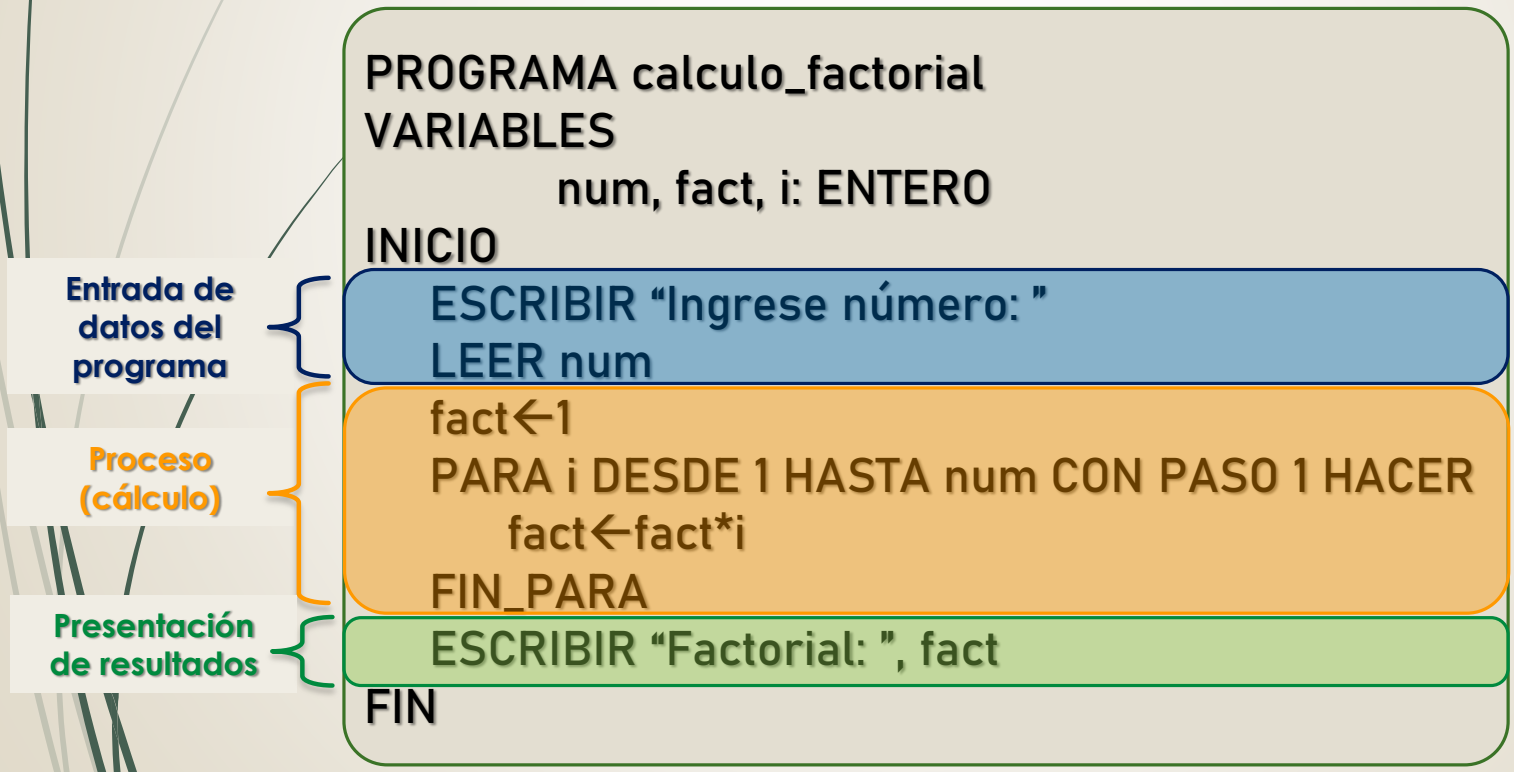
Ejemplo de Funciones (1)

- Ejemplo: Diseñe un programa modular que calcule el factorial de un número ingresado por el usuario.



Ejemplo de Funciones (2)

- Ejemplo: Diseñe un programa modular que calcule el factorial de un número ingresado por el usuario.



Ejemplo de Funciones (3)

- Ejemplo: Diseñe un programa modular que calcule el factorial de un número ingresado por el usuario.

```
PROGRAMA calculo_factorial
VARIABLES
   num, fact: ENTERO
```

```
FUNCIÓN factorial (E n: ENTERO): ENTERO
VARIABLES
     i,f: ENTERO
INICIO
     f←1
     PARA i DESDE 1 HASTA n CON PASO 1 HACER
          f←f*i
     FIN_PARA
     factorial←f
FIN
```

```
INICIO
     ESCRIBIR "Ingrese número: "
     LEER num
     fact←factorial(num)
     ESCRIBIR "Factorial: ", fact
FIN
```

Ejemplo de Funciones (4)

- Ejemplo: Diseñe un programa modular que calcule el factorial de un número ingresado por el usuario.

```
#include <iostream>
using namespace std;
```

```
int factorial(int n);
```

```
main()
{ int num, fact;
     cout << "Ingrese número: ";
     cin >> num;
     fact=factorial(num);
     cout << "Factorial: " << fact << endl;
     system("pause");
}
```

```
int factorial (int n)
{ int i,f;
  f=1;
  for(i=1;i<=n;i=i+1)
       f=f*i;
  return f;
}
```

Ejemplo de Funciones (5)

- Ejemplo: Diseñe un programa modular que calcule el producto (mediante sumas) de 2 números ingresados por el usuario.

```
#include <iostream>
using namespace std;
```

```
int producto(int a, int b);
```

```
main()
{ int num1, num2, prod;
      cout << "Ingrese número: ";
      cin >> num1;
      cout << "Ingrese número: ";
      cin >> num2;
      prod=producto(num1,num2);
      cout << "Producto: " << prod << endl;
      system("pause");
}
```

```
int producto(int a, int b)
{ int i,p=0;
  for(i=1;i<=b;i=i+1)
       p=p+a;
  return p;
}
```

Bibliografía

- Sznajdleder, Pablo Augusto. Algoritmos a fondo. Alfaomega. 2012.
- López Román, Leobardo. Programación estructurada y orientada a objetos. Alfaomega. 2011.
- De Giusti, Armando *et al*. Algoritmos, datos y programas, conceptos básicos. Editorial Exacta, 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.
- Joyanes Aguilar, Luis. Programación en Turbo Pascal. Mc Graw Hill. 1990.