Elementos de Programación

DIIT
Departamento de Ingeniería e Investigaciones Tecnológicas
Universidad Nacional de La Matanza

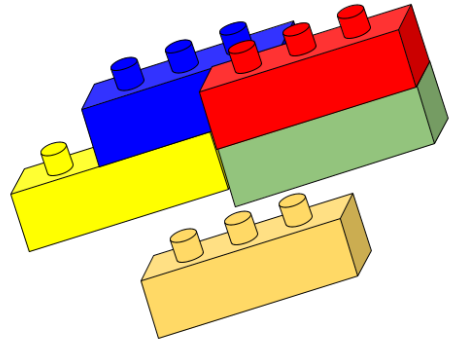
Elementos de Programación

Unidad 6: Funciones

Preparado por: Dr. Ing. Pablo Martín Vera

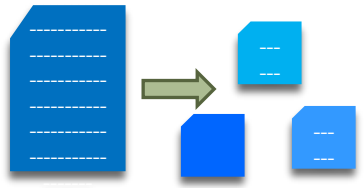
Funciones

- Una función es como un pequeño programa dentro del programa principal que cumple con una tarea particular
- Se las denomina también subprogramas, subrutinas o procedimientos

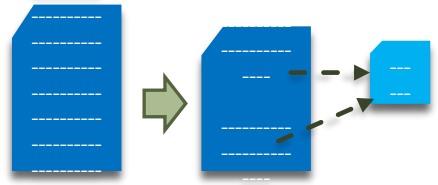


¿Para qué sirven las Funciones?

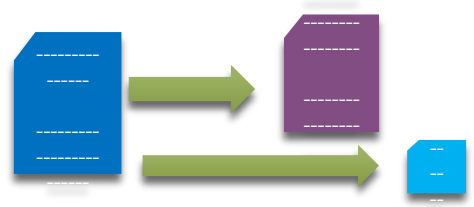
- Para dividir el programa en módulos separados facilitando su desarrollo y mantenimiento.



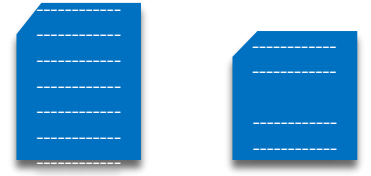
- Para escribir menos código. Ya que una función se puede utilizar todas las veces que se desee dentro de un programa.



- Para reutilizar código en los distintos programas



- Para ahorra memoria, ya que reduce el tamaño ocupado por el programa. No se escribe muchas veces el mismo código, sino que se invoca a la función que dentro ya tiene ese código



Funciones

- Pueden ser invocadas (hacer que se ejecuten) desde cualquier módulo o submódulo de un sistema. Es decir, que desde una función puede invocarse a otra sin ningún inconveniente.
- Opcionalmente pueden recibir datos que son enviados por el módulo que se invoca. Esto permite realizar funciones genéricas que se pueden reutilizar en varios casos.
- Opcionalmente pueden retomar un resultado al módulo que la invoca (este resultado será único valor).
- Cada función maneja un espacio de memoria diferente, por lo que las variables definidas en otros módulos no se pueden acceder.
- Al finalizar la función, las variables que se utilizaron internamente en la misma desaparecen ya que la memoria se libera y se crea un área distinta de memoria si se la vuelve a invocar nuevamente.

Componentes de una función

Toda función está compuesta por:

- El nombre o identificador
- La información que recibe (opcional)
- Las instrucciones que realizan la tarea deseada
- EL valor de retorno (opcional)

Declaración de una función - Prototipo

[void ó tipo] nombre ([parámetros formales con tipos ó solo el tipo de datos]);

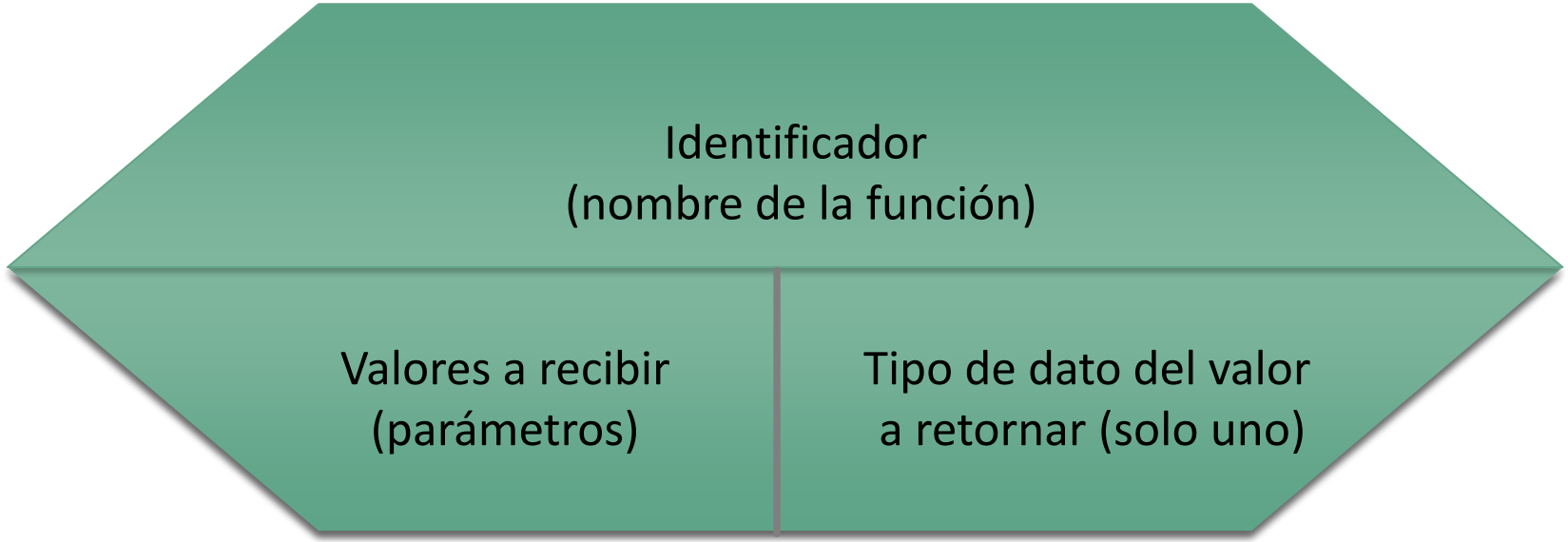
Ejemplos :

```
int   suma (int, int, int);

void  CalcularYMostrarPromedio(float, int);

char  IngresarVocal();

int   IngresaNumeroPar(void);

void  MostraTitulo();
```

Diagramación de Funciones

Al ser módulos independientes para cada función se va a desarrollar un diagrama separado.

Gráfico de inicio de una función

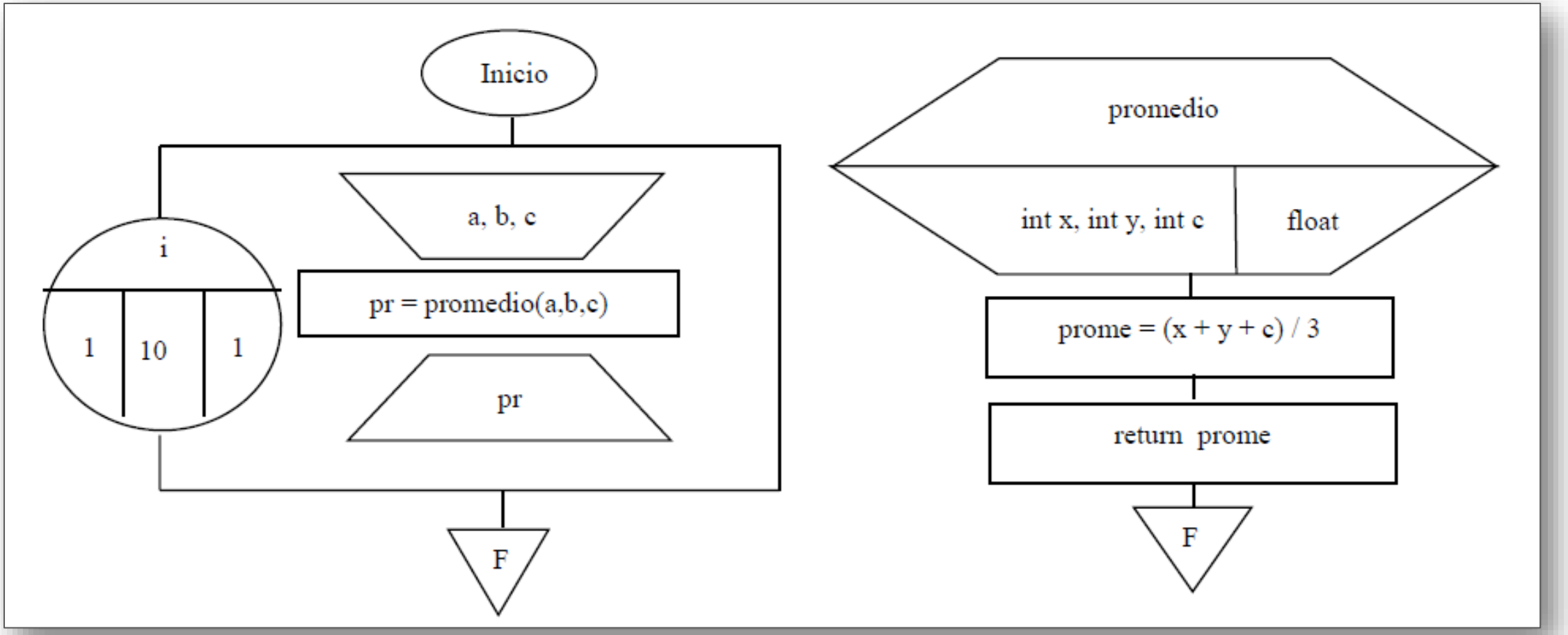


Diagramación de Funciones

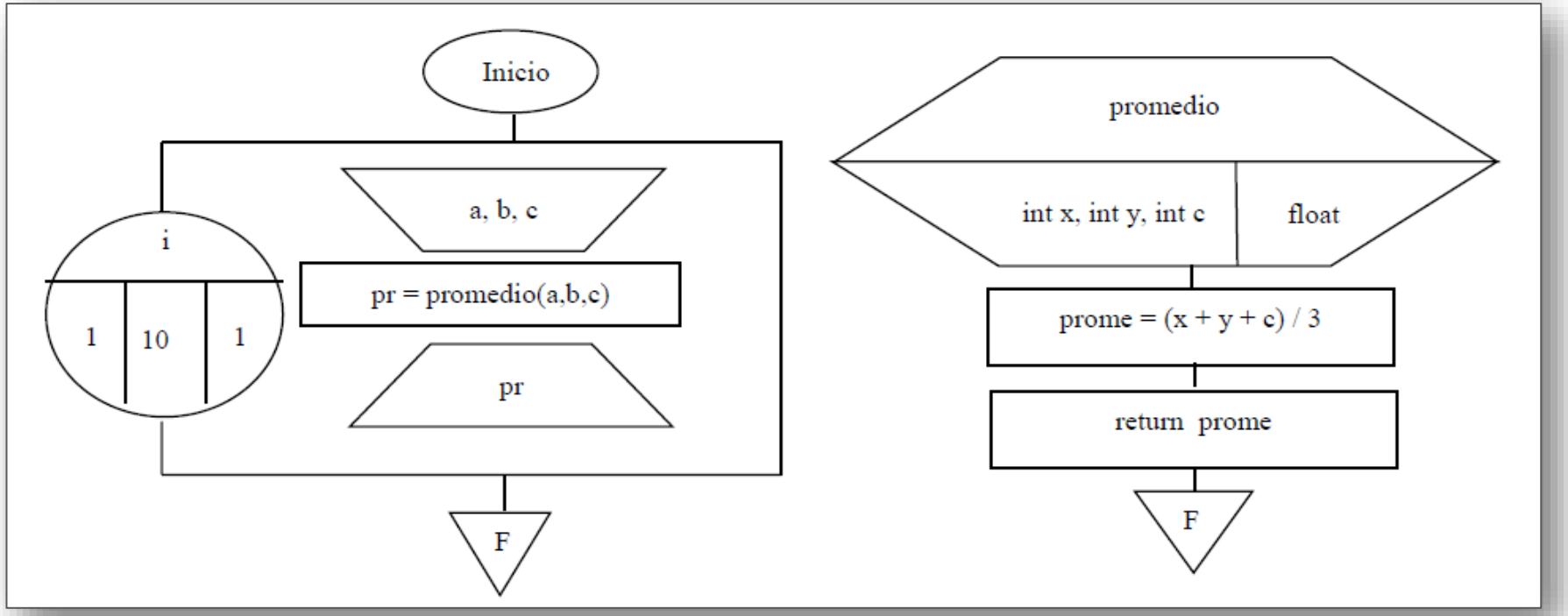
- Para invocar a una función se utiliza el grafico rectangular de proceso
- La función se invoca con su nombre y entre paréntesis los argumentos que se le envían, sino se envía nada se ponen los paréntesis vacíos
- Dentro de una función se utiliza la palabra reservada **return** para retornar un valor al módulo que la invoca, esta palabra hace que se salga de la función por lo que debe ponerse al final y en un único punto para respetar las reglas de la programación estructurada.

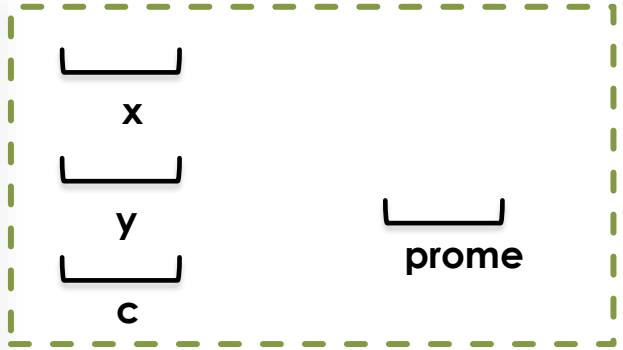
Ejemplo de utilización de una Función

Este programa ingresa 10 ternas de valores y calcula el promedio de cada terna.

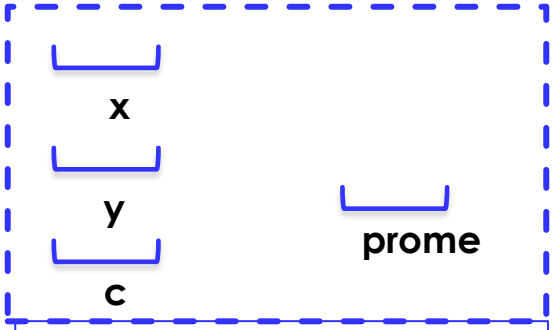


Áreas de memorias separadas – Variables locales

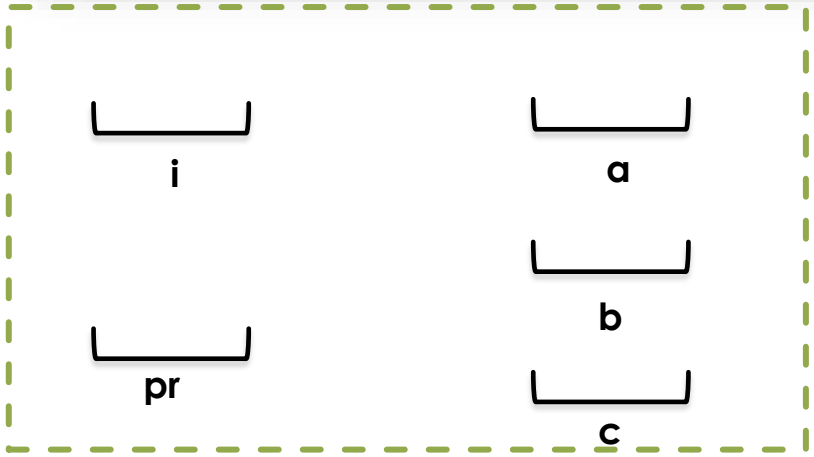




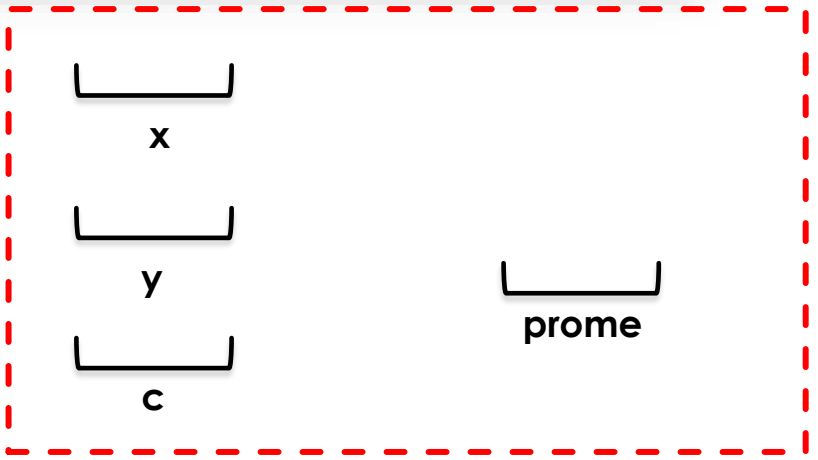
Área de memoria de la función promedio
Distinta en cada invocación



Área de memoria de la función promedio
Distinta en cada invocación



Área de memoria del programa principal



Área de memoria de la función promedio
Distinta en cada invocación

Codificación de Funciones

Tipo de dato que retorna o void sino retorna nada

Nombre / Identificador de la función

```
#include <stdio.h>
#include <stdlib.h>
```

PROTOTIPO

```
float  promedio  ( float , float , float);
```

Tipo de datos de los argumentos que recibe, o void o paréntesis vaciós sino recibe nada

LLAMADA O INVOCACIÓN

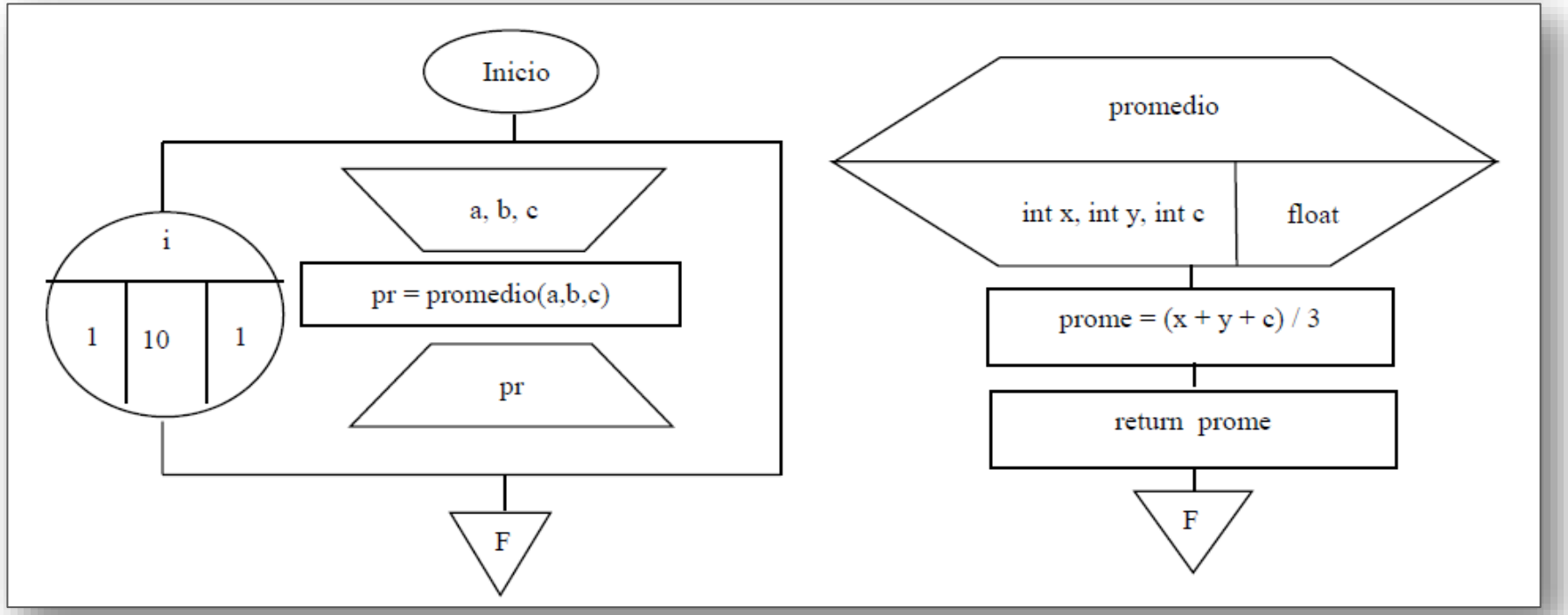
```
int main()
{
   .....
    pr = promedio( a , b , c);
 .....
  return 0;
}
```

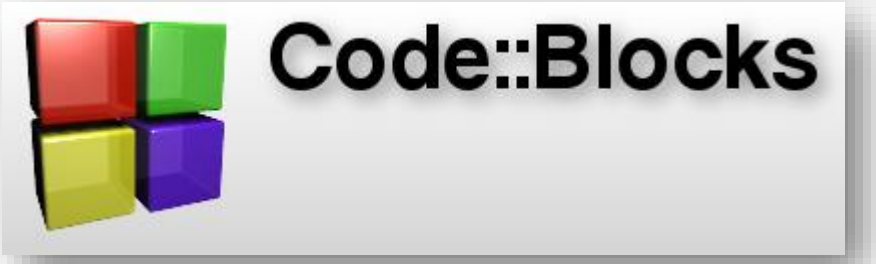
DESARROLLO

```
float   promedio ( float  x ,  float   y,  float  c)
{
        //declaración de variables locales
        //código de la función

}
```

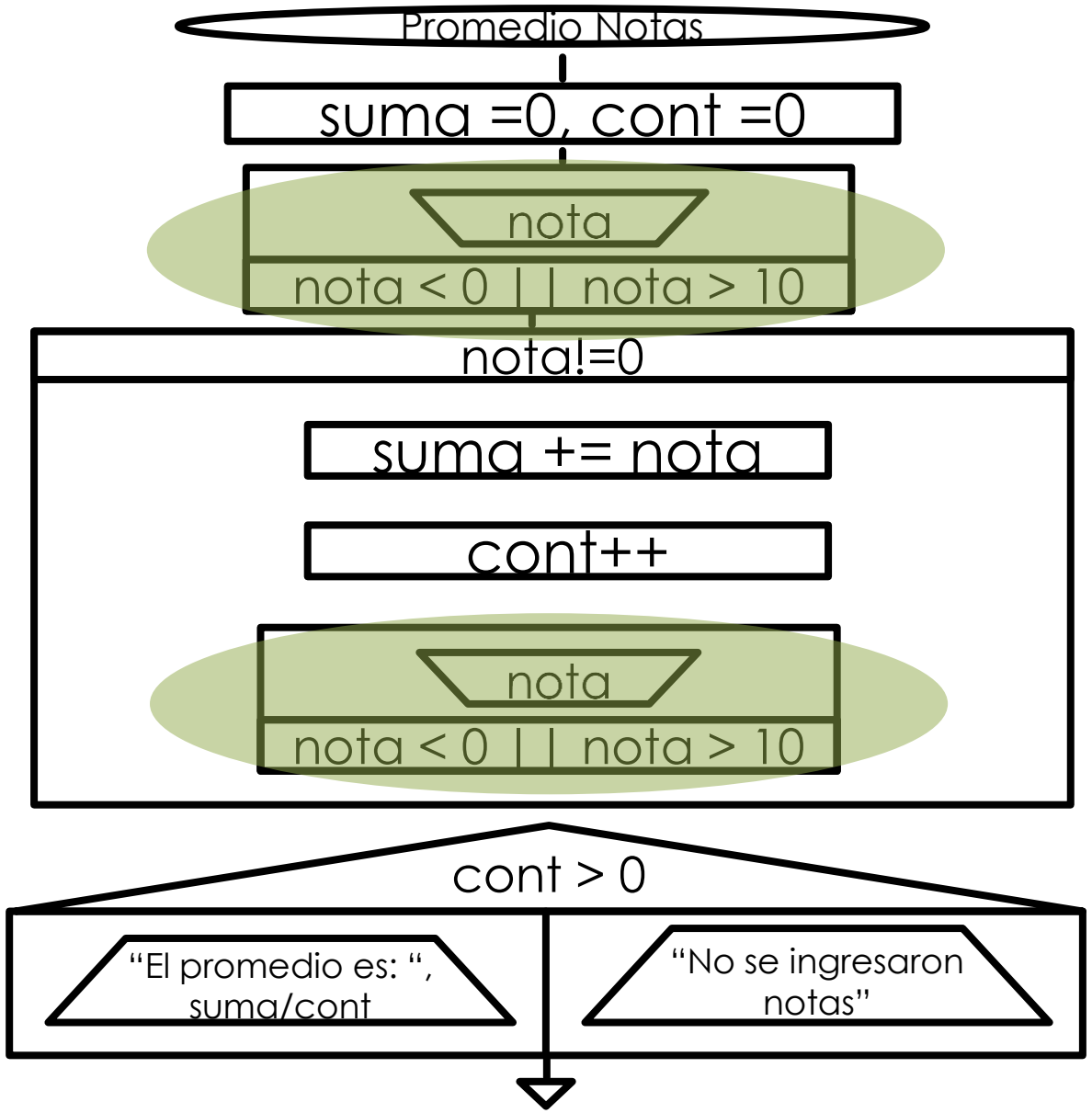
Por cada parámetro formal se crea automáticamente una variable local NO SE DEBE volver a declarar

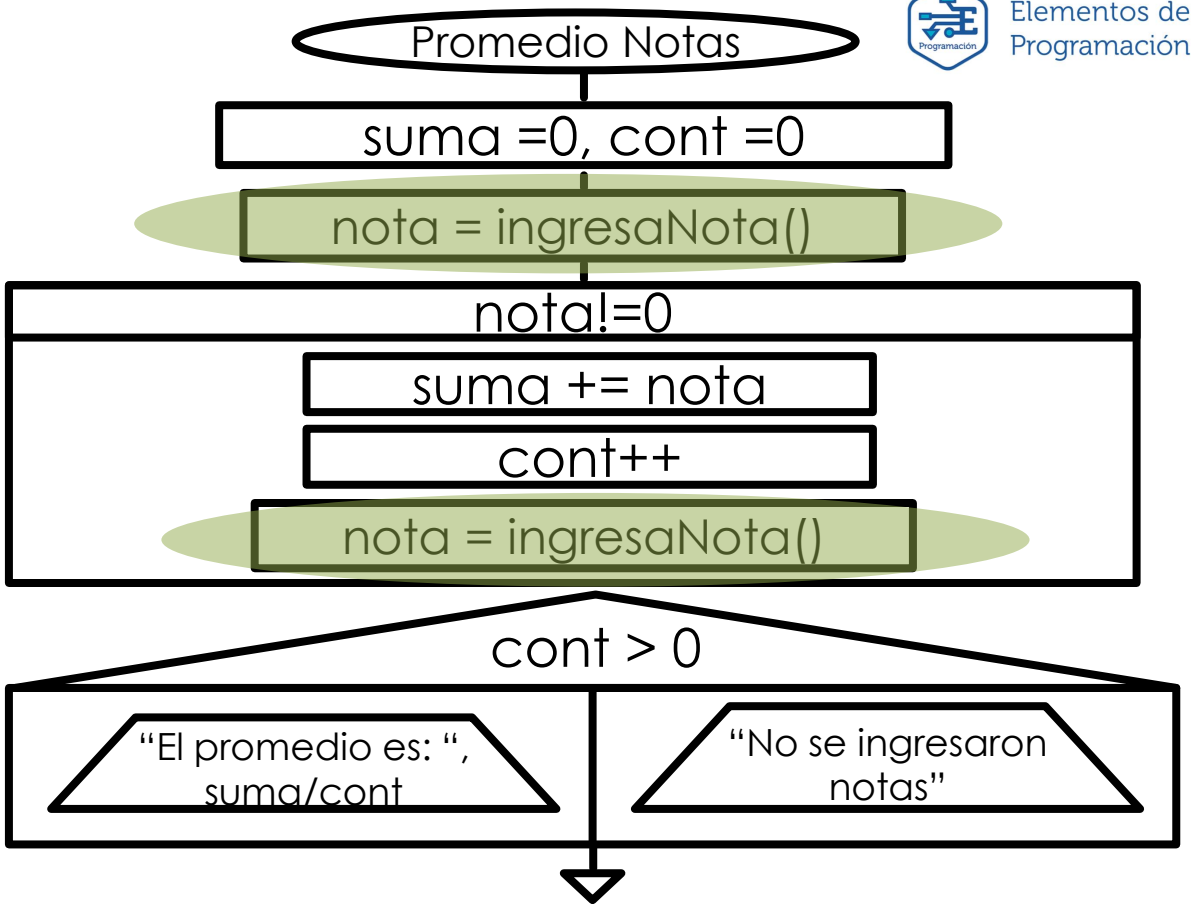
Codificamos la función

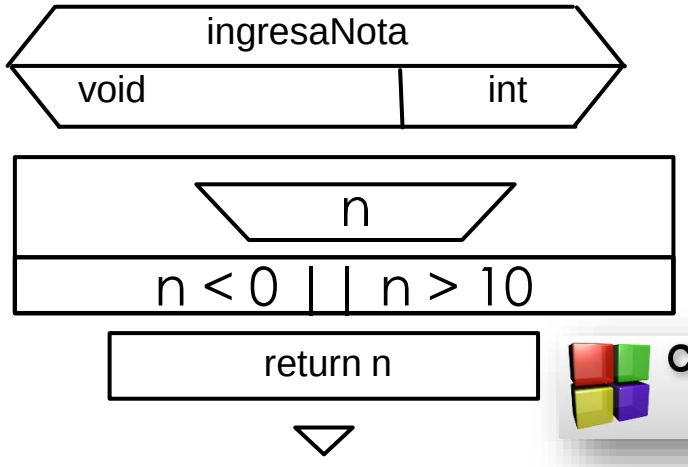




Funciones para reutilizar código

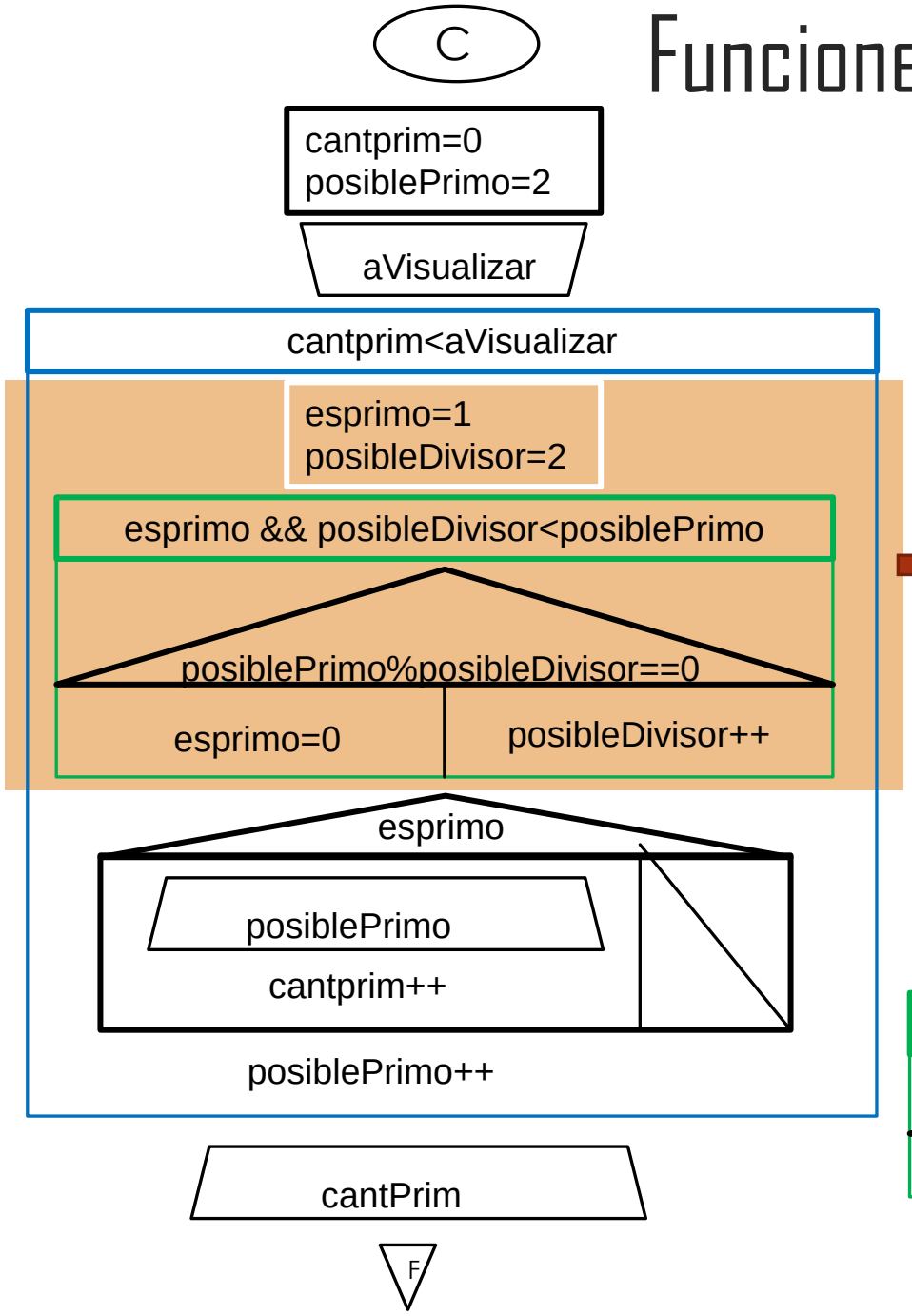




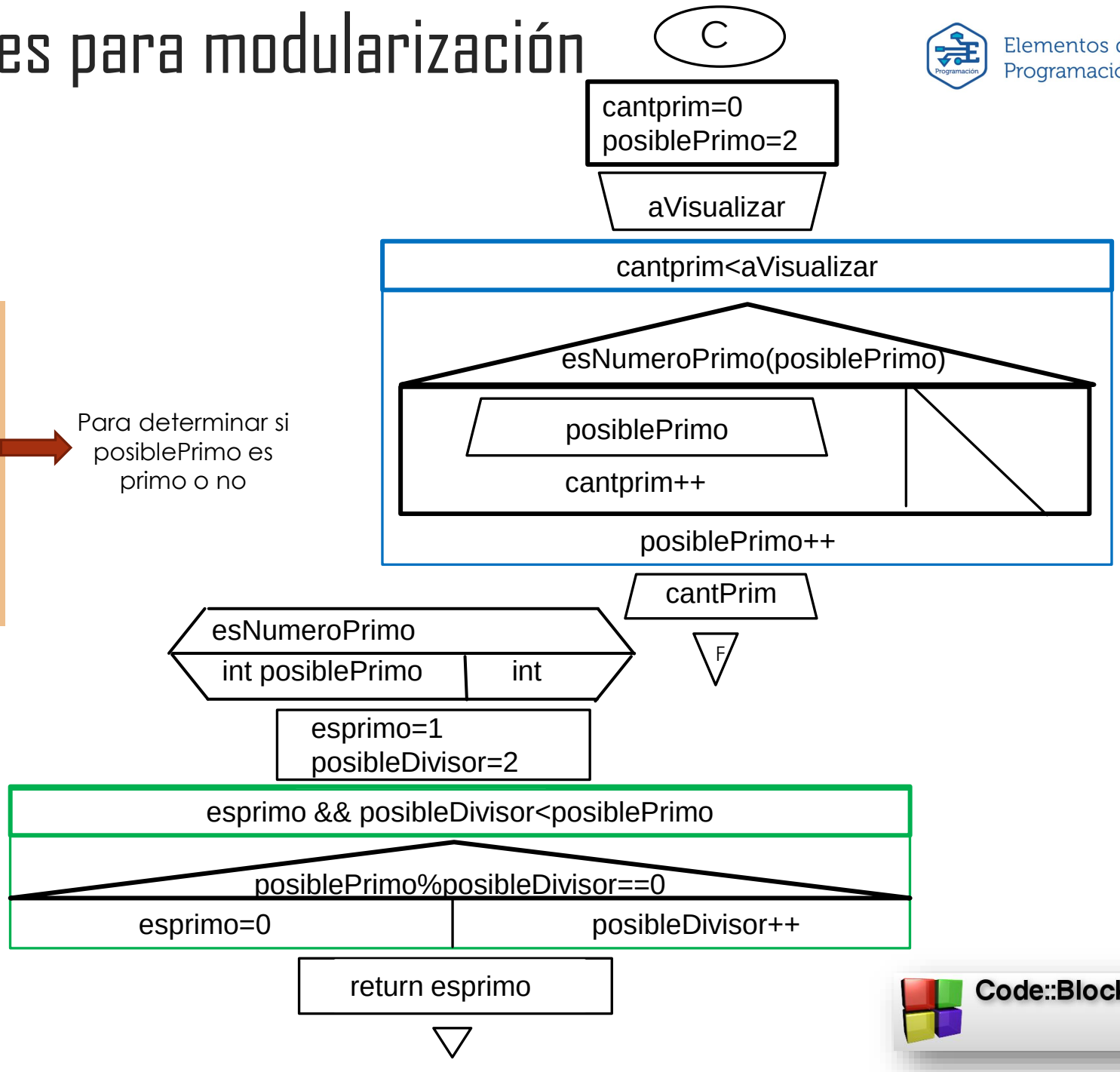


Code::Blocks

Funciones para modularización



Para determinar si posiblePrimo es primo o no



Code::Blocks

Metodologías para la programación modular

La programación modular permite separar el problema en subproblemas menores. Es decir, que parte de la lógica se puede delegar para que sea resuelta por una función lo que permite encarar el diseño de un algoritmo de dos formas distintas:

Botton-Up (ascendente) – De lo particular a lo general

Esta metodología se basa en comenzar a diseñar el algoritmo desarrollando primero las funciones o viendo que funciones se tienen para luego con esas funciones armar el diseño del programa principal. Es decir, que parte de lo particular para luego generar una solución al problema.

Top-Down (descendente) – De lo general a lo particular

Se realiza un diseño general del sistema centrándose en la lógica del programa principal, delegando algunas tareas en funciones que serán desarrolladas luego. Es decir, que primero se hace un diseño general y luego se va a lo particular especificando las partes de la solución que falten.

Elementos de Programación

Resolver ejercicio 6.1

Preparado por: Dr. Ing. Pablo Martín Vera

DIIT
Departamento de Ingeniería e Investigaciones Tecnológicas
Universidad Nacional de La Matanza