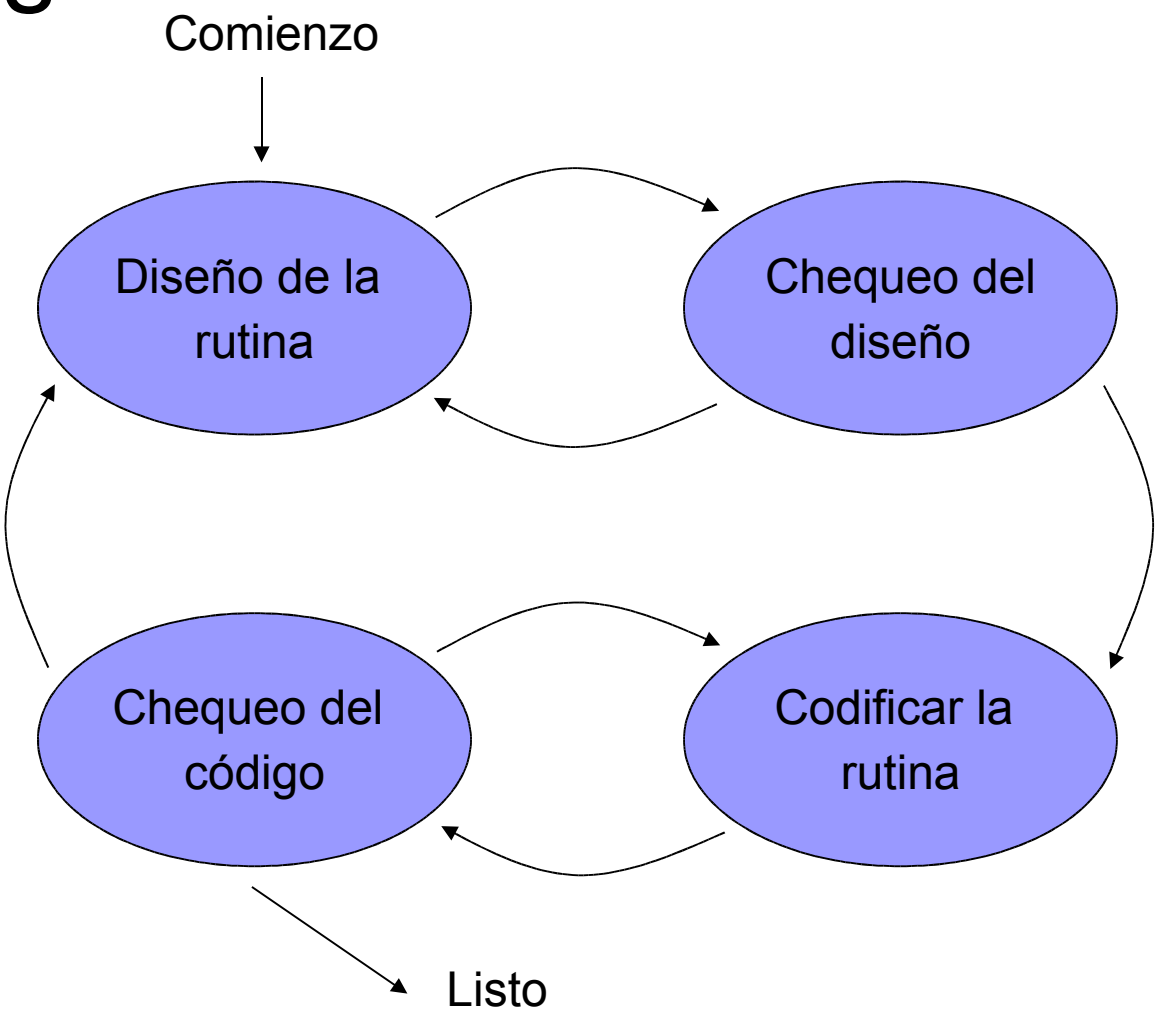
Funciones y Procedimientos I

Andrés Arcia
Departamento de Computación
Escuela de Ingeniería de Sistemas
Facultad de Ingeniería
Universidad de Los Andes

Pasos generales para construir rutinas



Diseño de una rutina

Chequeo de prerrequisitos. Chequear bien si el trabajo de la rutina está bien definido.

Definir el problema de la rutina en términos de los datos de entrada, salida y manejo de errores.

Nombre de la rutina. Aunque parezca trivial, solo los buenos nombres dan señal de buenos programas. El nombre debe ser conciso y expresivo. No debe dejar duda de su funcionalidad que la rutina ofrece.

Diseño de una rutina

Proponga estrategias de prueba para la rutina. Por ejemplo, para calcular el área de un triangulo dado la base y la altura, deben chequearse si la base o la altura son cero, negativas, etc.

Itere. Intente muchas ideas para codificar la misma rutina. La construcción de programas es iterativo, es decir, se intenta, se falla y se vuelve a intentar hasta conseguir la versión definitiva.

Codificación de una rutina

El diseño de una rutina es tan importante como el plano de una casa antes de construirla o, la planificación de una ruta antes de una excursión.

Una vez que ya ha diseñado la rutina, entonces implante.

Codificación de una rutina

Escriba la declaración de la rutina. Esto se conoce comúnmente como interfaz (entre dos caras).

Escriba el algoritmo con frases de muy alto nivel.

Complete cada frase de alto nivel con una o más líneas de código.

Chequee el código informalmente. Haga una prueba de los trozos de código que se van añadiendo debajo de los comentarios.

Chequeo formal del código

Haga una inspección mental del código. Repase los algoritmos que se han inventado para la solución del problema. Explicar el código a otros, también ayuda a la comprensión.

Compile la rutina. El compilador dirá si todos los constructos están bien escritos y tienen sentido. Algunas recomendaciones son: encender todos los niveles de advertencia del compilador y siempre elimine todas las causas de error y advertencias. (-w para inhibir todos los mensajes, -Wall y –W para encender la mayoria de las advertencias).

Chequeo formal del código

Utilice un depurador para hacer un seguimiento minucioso a su programa.

Nunca deje pasar un error por pequeño que sea.

Piense siempre en rutinas antes de repetir código.

Módulos

Son un conjunto de rutinas que prestan un servicio especifico. Hay criterios para establecer el conjunto, que siempre debería ser completo y ¡perfecto! (Vea la definición de perfección de Antoine de Saint-Exupéry)

Una rutina también puede ser considerada como módulo, siempre y cuando no existan otras rutinas afines a las cuales pueda unirse.

Modularidad: Cohesión y Acoplamiento

La modularidad es un compromiso entre el diseño de rutinas y el diseño de módulos.

La modularidad da una mejor comprensión del problema y reduce el tamaño del código.

Es bueno pensar en módulos como cajas negras. Se sabe que entra y que sale, pero no como se procesa.

Cohesión

Un módulo debe ofrecer un grupo servicios que sin lugar a dudas deben ir juntos. (math.h, stdio.h, etc).

En la implantación de un navegador *web*, todos los módulos deben tener como objetivo cooperación mutua para la utilización de recursos compartidos. Para ello, el navegador está compuesto de un módulo de comunicación, uno para el manejo de ventanas, otro para el manejo de archivos, etc.

Acoplamiento

Un módulo debe ofrecer una colección de servicios diseñados para que el resto del programa pueda interactuar con él.

En el navegador, el modulo de comunicación debe tener las rutinas: establecerConexion(), cortarConexion(), obtenerImagen(), obtenerMP3().

¿Por qué Módulos?

Interfaz de usuario. A través de los módulos las interfaces se pueden independizar.

Código reutilizable. Se puede evitar rescribir código.

Operaciones relacionadas son propensas a cambiar.

Sugerencias

La diferencia entre rutina y módulo es importante.

Se pueden implantar módulos en cualquier lenguaje. En caso de no proveerlo, puede utilizar constructos para alcanzar un nivel de modularización. Ej. Las clases en C++ o java, Bibliotecas de C, etc.

Ventajas

Como los módulos son independientes, varios programadores pueden trabajar simultáneamente y de manera independiente en la resolución de un problema, repartiéndose las distintas partes del mismo.

Facilita la escritura y depuración de un programa ya que cada módulo representa una parte bien definida del problema.

Localización rápida de errores.

Modificación de un módulo sin afectar a los demás.

Ventajas

Un grupo de instrucciones que se repite en distintas partes de un programa se pueden incluir dentro de una sola función, a la que se puede utilizar siempre que sea necesario con datos de entrada adecuados para su uso en un contexto particular y desde distintas partes del programa.

Favorece la portabilidad, ya que se pueden escribir programas sin prestar atención a las características de un sistema en particular.

Programación Modular

Diseño descendente (de arriba hacia abajo): En la solución de problemas grandes, es conveniente dividirlo en problemas mas pequeños (sub-problemas), los cuales a su vez pueden dividirse en sub-problemas más pequeños.

Este es un proceso de refinamiento por pasos, etapas o capas. Comenzando desde lo más general hasta lo más especifico.

Sobre el Diseño Descendente

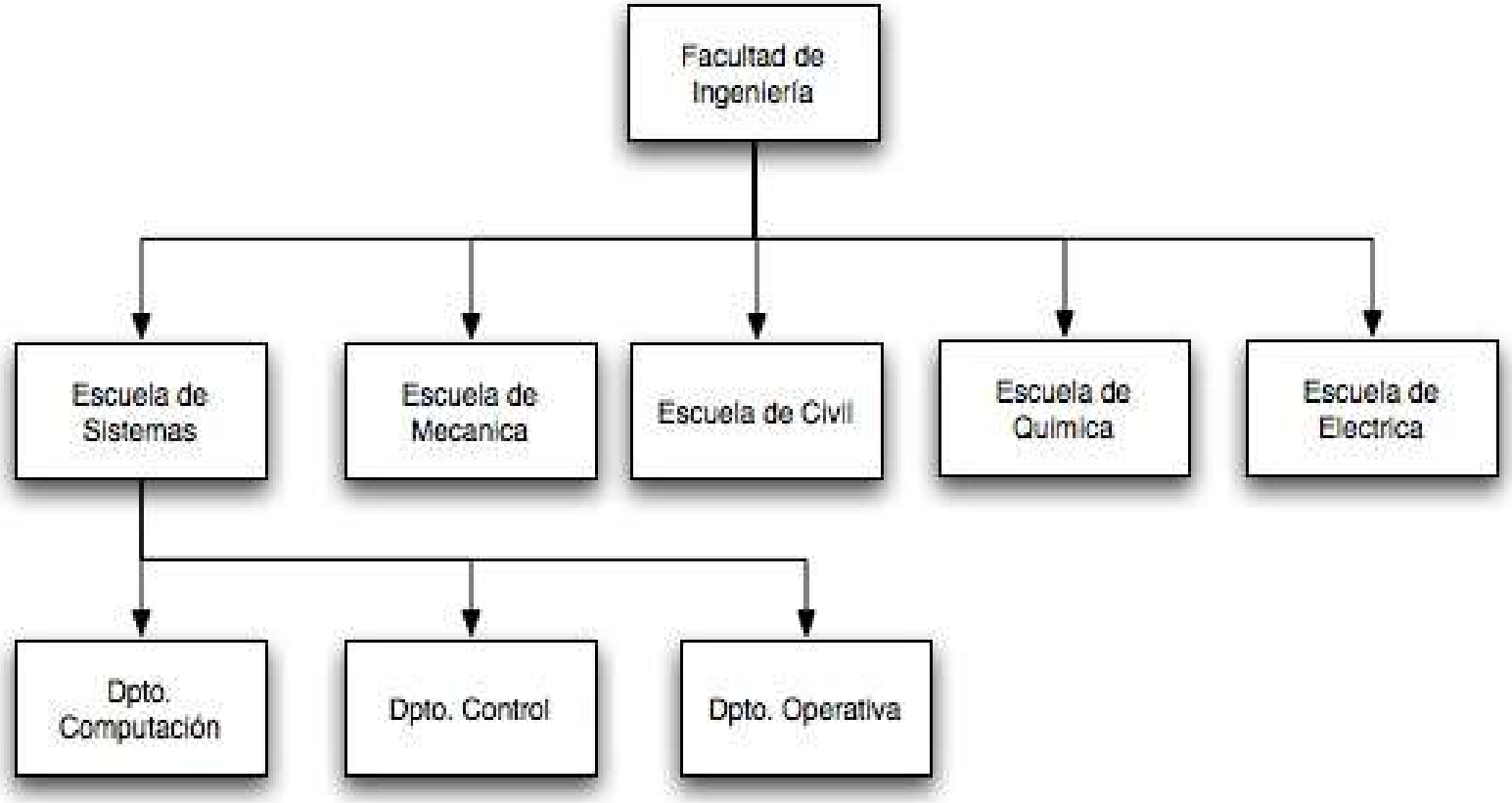
Comience siempre por lo más general, luego dividir y vencer…

Deje claras las dependencias del lenguaje. Esto permite cambiar de lenguaje en medio del diseño sin ningún trauma.

Postergue lo más posible el trabajo en los detalles.

Verifique cuidadosamente cada nivel.

Abordaje Modular



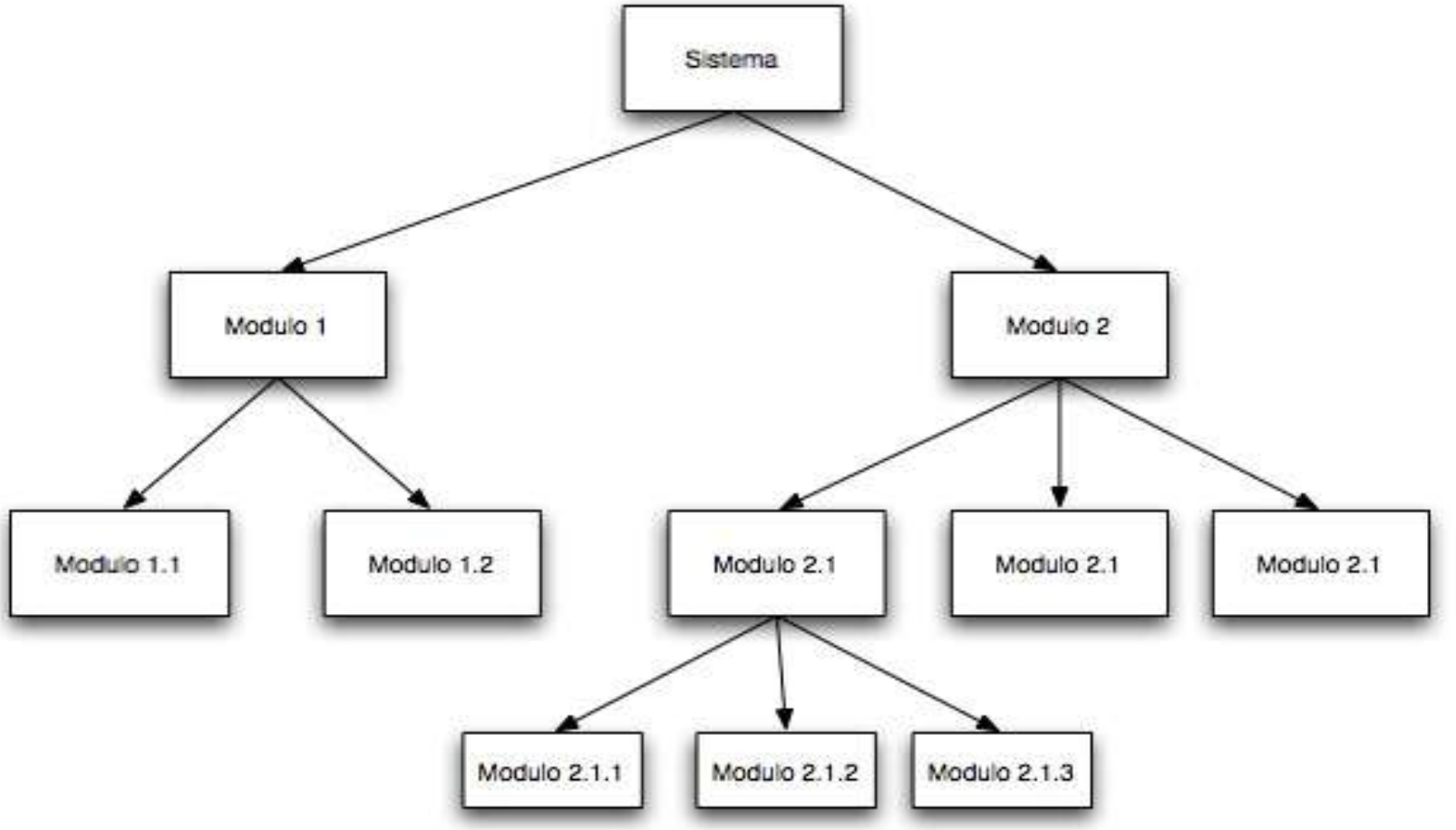
Programación Modular

Rutinas: Piezas pequeñas diseñadas para ejecutar una tarea específica. Permiten desarrollar y mantener programas grandes y módulos. Pueden ser de dos tipos:

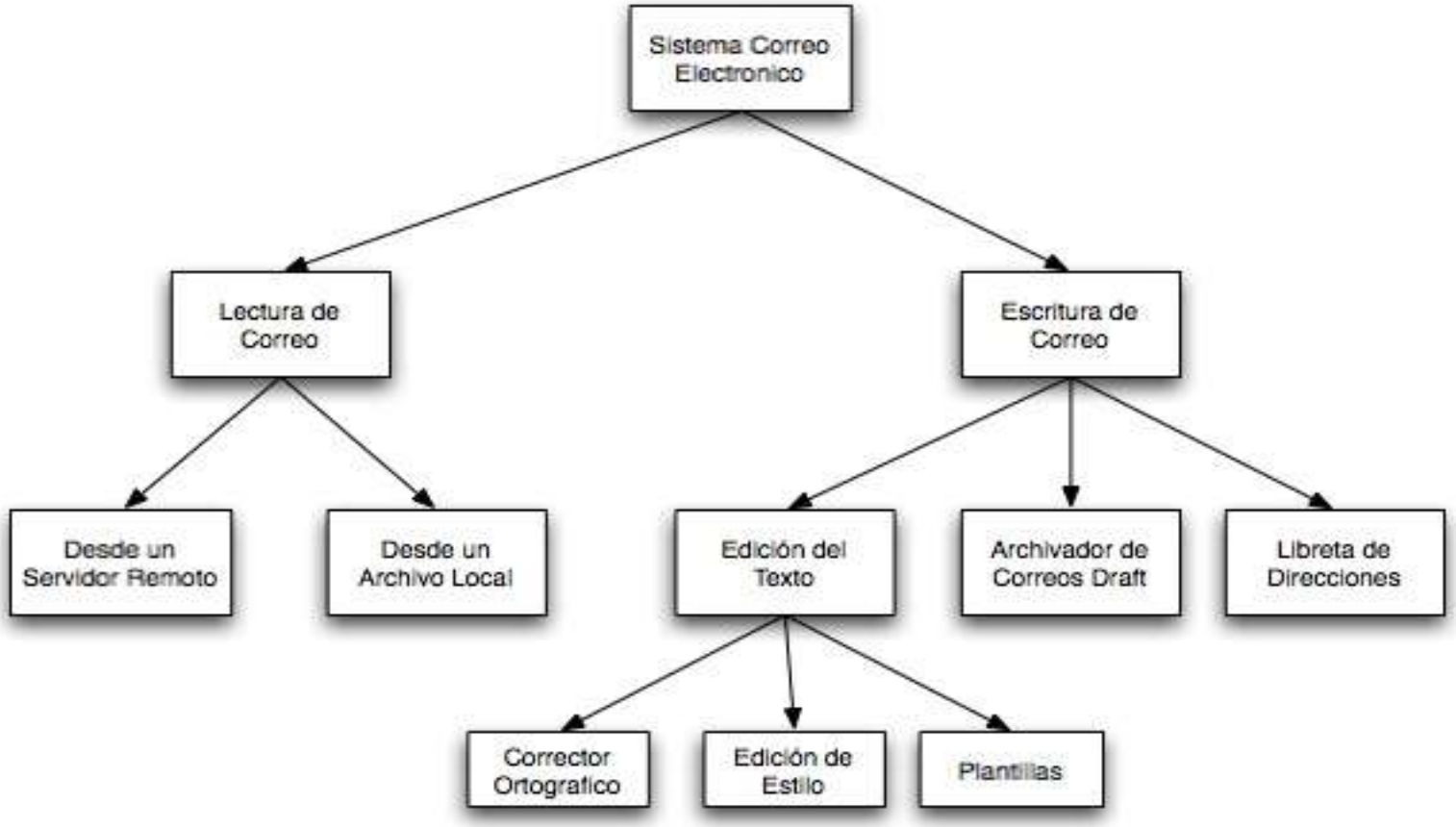
- Funciones
- Procedimientos

Programación Modular: Método de resolución de problemas que consiste en resolver de forma independiente los sub-problemas resultantes de una descomposición. En la programación módular deben asegurarse los preceptos de máxima cohesión y mínimo acoplamiento entre los módulos.

Programación Modular



Programación Modular: Ejemplo 1



Programación Modular: Ejemplo 2

Módulo de Calculo del área de un rectángulo.

Sub-problemas:

- Entrada de datos de la altura y la base.
- Calcular la superficie.
- Salida de los resultados.

Rutinas:

- LeerDatos(altura, base)
- CalcularArea(altura, base, area)
- EscribirArea(area)

Reutilización de un módulo

Los algoritmos de cada módulo sólo se escriben y codifican una sola vez, aunque se necesiten en distintas ocasiones a lo largo del programa completo e incluso de otros programas (reutilización) evitando la duplicación innecesaria de código

La reutilización de un módulo por otros programas es un ahorro de tiempo, ya que no es necesario volver a resolver el problema, y si el módulo ha sido previamente probado y verificado también reduce la posibilidad de errores.

Fácil comprensión del programa completo.

Función

Matemáticamente una función es una **operación** que toma uno o mas valores llamados **argumentos** y produce un valor llamado **resultado**.

Ejemplos

$f(x) = \dfrac{4}{x^2 + 1}$ función de un sólo argumento

$f(4) = 4/(16 + 1) = 4/17 = 0.235$

$f(a, b) = \dfrac{a^2 + b^2}{(a - 1)^2 - (b - 1)^2}$ función de dos argumentos

$f(2, 3) = (4 + 9)/(1 - 4) = -13/3 = -4.33$

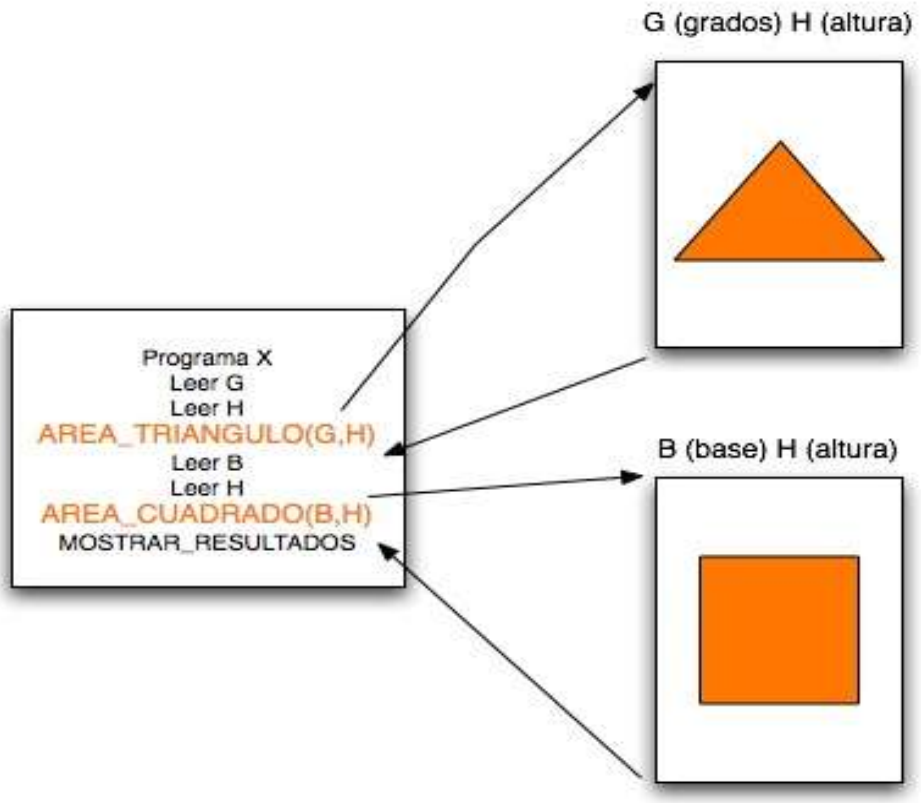
Función

En C los módulos se llaman funciones (unidad básica de los programas). Realizan determinadas tareas bien definidas.

Una función tiene un *nombre*, toma cero o mas valores, denominados *argumentos* o *parámetros de entrada* y, según el valor de éstos, devuelve un *resultado*, en el nombre de la función, el cual es obtenido durante su ejecución.

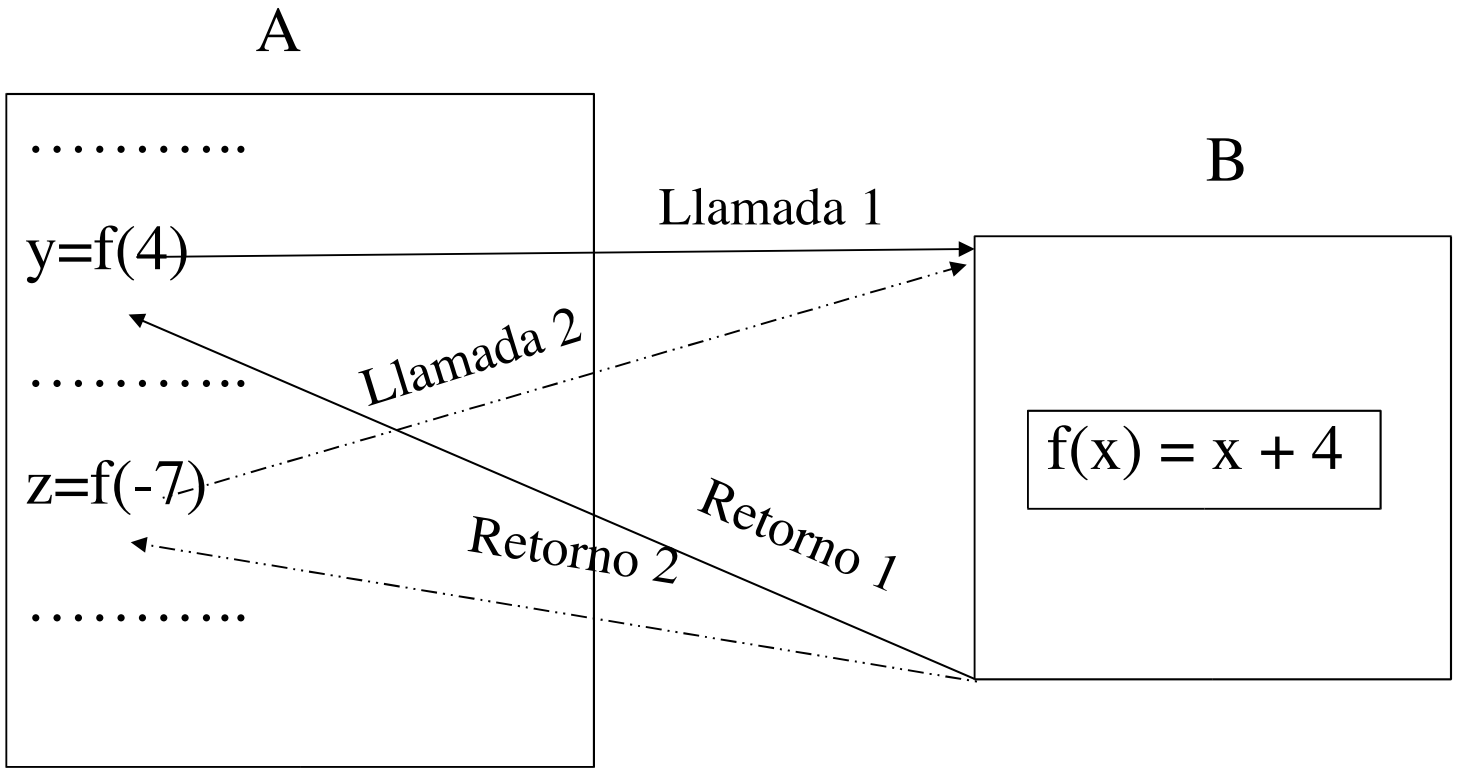
Una función se define una sola vez pero puede usarse (mediante llamadas) tantas veces como sea necesario.

Mecanismo de Llamadas entre Funciones



Nota: Cada vez que una función es llamada, se ejecuta y retorna el control al lugar desde donde fue hecha la llamada.

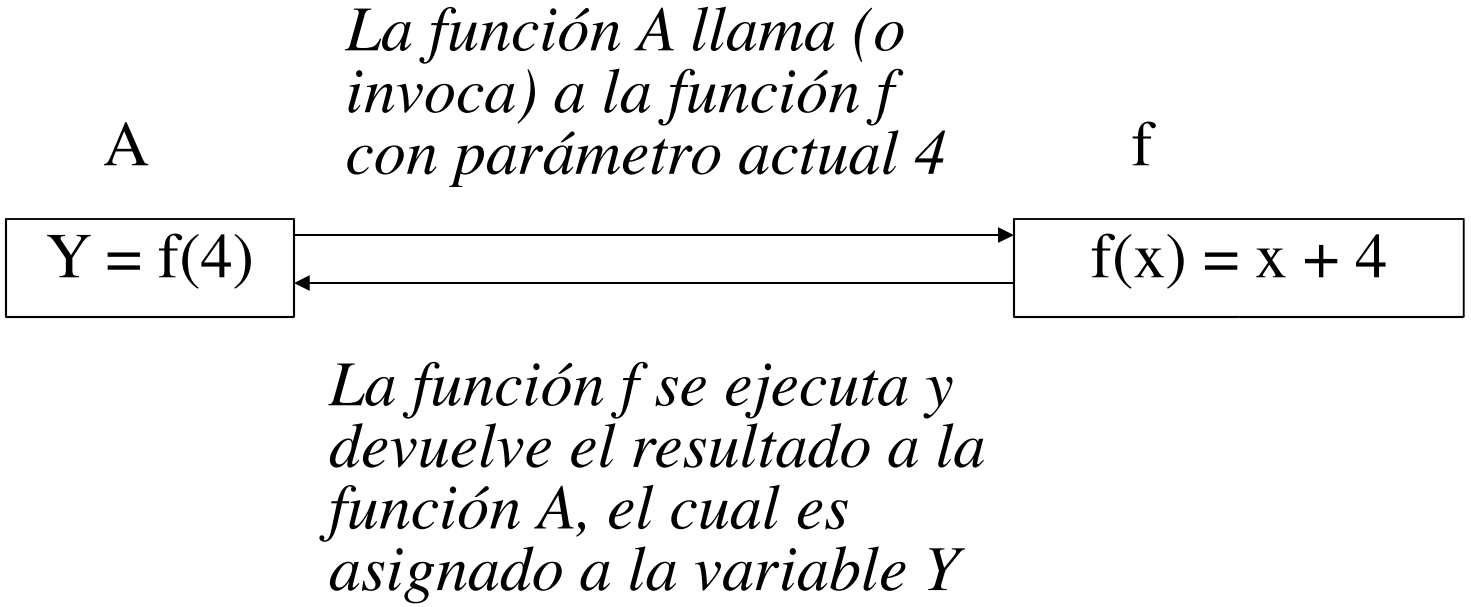
Mecanismo de Llamadas entre Funciones



Nota: Cada vez que una función es llamada, se ejecuta y retorna el control al lugar desde donde fue hecha la llamada.

Mecanismos Utilizados para pasar Información entre Funciones

Parámetros: Proporcionan la forma de comunicar información entre funciones.



Tipos de Funciones en C

Funciones de biblioteca (módulos): C tiene sus propio set de bibliotecas de funciones que permiten realizar ciertas operaciones o cálculos de uso común.

Funciones definidas (diseñadas y codificadas) por el programador para realizar determinadas tareas.

Biblioteca Estándar de C

Contiene una amplia colección de funciones para llevar a cabo cálculos matemáticos comunes, manipulaciones con cadenas de caracteres, manipulaciones con caracteres, operaciones de entrada/salida y muchas otras operaciones útiles.

Esta biblioteca de funciones comunes construida una vez, puede ser reutilizada por diferentes programas.

La biblioteca estandar

- Entrada y salida estandar: stdio.h
  - Operaciones sobre archivos
  - Entrada/Salida formateada
  - Entrada/Salida de caracteres
- Pruebas para caracteres: ctype.h
- Cadenas: string.h
- Funciones matematicas: math.h
- Funciones utilirarias: stdlib.h (atoi, atof, rand, srand, calloc, malloc, realloc, free, exit)

La biblioteca estandar

Diagnostico: assert.h

Fecha y Hora: time.h

Limites de los tipos de datos: limits.h

Funciones de Biblioteca mas usadas en C

FUNCION	TIPO	PROPOSITO	ARCHIVO *include*
abs(i)	int	devuelve el valor absoluto de i	stdlib.h
fabs(d)	double	devuelve el valor absoluto de d	stdlib.h
cos(d)	double	devuelve el coseno de d	math.h
cosh(d)	double	devuelve el coseno hiperbólico de d	math.h
exp(d)	double	devuelve el valor e^d	math.h
log(d)	double	devuelve el logaritmo natural de d	math.h
sqrt(d)	double	devuelve la raíz cuadrada de d	math.h
floor(d)	double	devuelve el entero mas grande no mayor que d	math.h
ceil(d)	double	devuelve el entero mas pequeño mayor o igual a d	math.h
pow (d1, d2)	double	devuelve d1 elevado a la potencia d2	math.h
fmod(d1, d2)	double	devuelve el resto de d1/d2 con el mismo signo de d1	math.h

Funciones de Biblioteca mas usadas en C

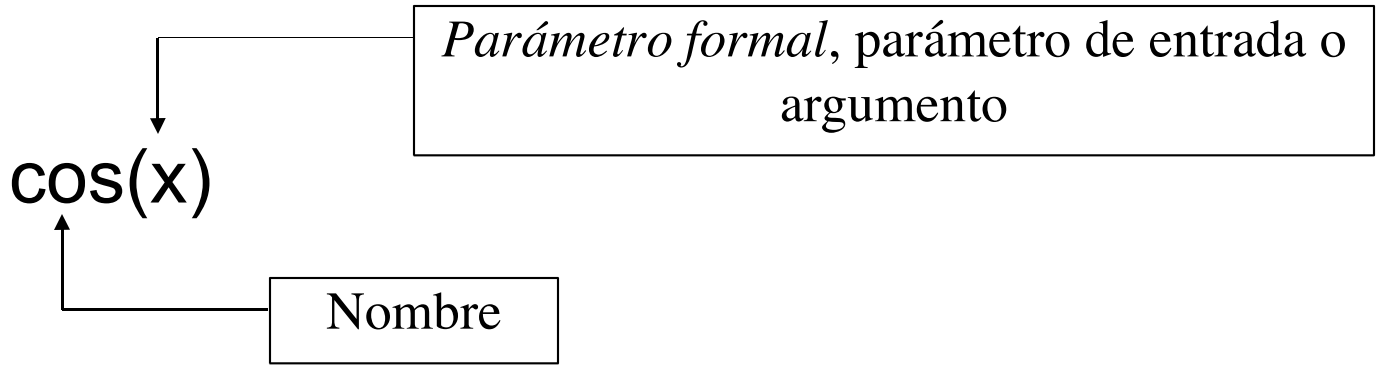
FUNCION	TIPO	PROPOSITO	ARCHIVO *include*
sin(d)	double	devuelve el seno de d	math.h
acos(d)	double	devuelve el arco coseno de d	math.h
asin(d)	double	devuelve el arco seno de d	math.h
atan(d)	double	devuelve la arco tangente de d	math.h
atan(d1, d2)	double	devuelve la arco tangente de d1/d2	math.h
atof(s)	double	convierte la cadena s a una cantidad en doble precisión	stdlib.h
atoi(s)	int	convierte la cadena s a un entero	stdlib.h
atol(s)	long	convierte la cadena s a un entero largo	stdlib.h
getchar()	int	lee un carácter desde el dispositivo de entrada estándar	stdio.h
putchar(c)	int	escribe un carácter en el dispositivo de salida estándar	stdio.h
tolower(c)	int	convierte una letra a minúscula	ctype.h
toupper(c)	int	convierte una letra a mayúscula	ctype.h

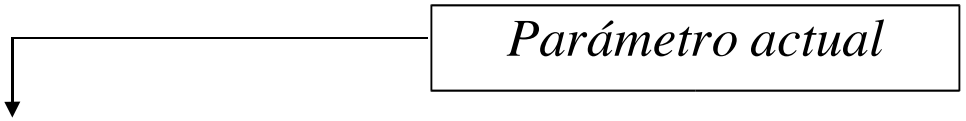
Funciones de Biblioteca mas usadas en C

Ejemplos

```
valor1 = sin(3.14159);
valor2 = sin(x) - cos(y);
valor3 = sin(theta)/(sin(delta) - sin(delta-theta));
theta = acos(1.0/sqrt(1 - x*x));
```

Funciones de Biblioteca mas usadas en C





Ejemplo:

valor1 = cos(3.14159);

Llamada a la función coseno de la biblioteca math.h

Tipos de Parámetros

Parámetros formales (parámetros de entrada o argumentos): Declaraciones de los parámetros en la definición de la función.

Parámetros actuales: Valores que toman los parámetros formales y que son proporcionados a la función que es llamada por la función que la llamó.

Funciones de Biblioteca: Ejemplo 1

Un polinomio de x, de segundo grado como máximo, se da por

$$ax^2 + bx + c$$

Su discriminante se define como

$$\sqrt{b^2-4ac}$$

Se desea conocer la raíz cuadrada del discriminante; si el discriminante no es negativo,

tiene la interpretación usual, pero si el discriminante es negativo, entonces

$\sqrt{b^2-4ac}$ significa $i*\sqrt{b^2-4ac}$

donde i es el número imaginario que satisface

$i^2=-1$ o en forma equivalente, $i=\sqrt{-1}$

Dado los valores de a, b y c, calcular el valor de la raíz cuadrada del discriminante.

Funciones de Biblioteca: Ejemplo 1

Análisis E-P-S

Entradas: a, b, c $\in$ R

Proceso:

- Calcular discriminante = $\sqrt{b^2-4ac}$
- Si discriminante $\geq$ 0 entonces RC = $\sqrt{b^2-4ac}$
- Si discriminante < 0 entonces RC = $i*\sqrt{b^2-4ac}$

Salidas: Raíz cuadrada del discriminante RC $\in$ R

Funciones de Biblioteca: Ejemplo 1

Algoritmo

0. Inicio
1. Escribir (“a = ?, b = ?, c = ?”)
2. Leer (a, b, c)
3. disc= pow (b, 2.0) - 4.0ac
4. Si (disc ≥ 0) entonces
    RC = sqrt(disc)
    Escribir (“Raiz del discriminante = “, RC)
        sino
    RC = sqrt(-disc)
    Escribir (“Raiz del discriminante = i“, RC)
  fin-si
5. Fin

Funciones de Biblioteca: Ejemplo 1

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

void main () {
    float a, b, c, disc, RC;

    printf("Valores de a = ?, b = ?, c = ? \n");
    scanf("%i%i%i",&a , &b, &c);
    disc= pow (b, 2.0)  + 4.0*a*c;
    if (disc >= 0.0) {
        RC = sqrt(disc);
        printf("Raiz del discriminante = %f", RC);
    } else {
        RC = sqrt(-disc);
        printf("Raiz del discriminante = i%f", RC);
    } }
```

Funciones de Biblioteca: Ejemplo 2

Leer una letra minúscula y convertirla a mayúscula.

Análisis E-P-S

Entradas: Letra minúscula

Proceso:

- Convertir la letra minúscula en su correspondiente mayúscula

Salidas: Letra mayúscula

Funciones de Biblioteca: Ejemplo 2

Algoritmo

```
0. Inicio
1. Escribir ("Introduzca una letra minúscula")
2. Leer (minuscula)
3. mayuscula = convertirMayuscula(minuscula)
4. Escribir (mayuscula)
5. Fin
```

Funciones de Biblioteca: Ejemplo 2

Codificación

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

int main()
{
        int minuscula, mayuscula;

        printf("Introduzca una letra minúscula");
        minuscula = getchar();
        mayuscula = toupper(minuscula);
        putchar(mayuscula);
}
```

Ejercicios

Un programa en C contiene las siguientes declaraciones y asignaciones iniciales:

```
int i = 8, j = 5;
double x = 0.004, y = -0.01;
char c = 'c', d = 'd';
```

Determinar el valor de cada una de las siguientes expresiones, que hacen uso de funciones de biblioteca.

a) abs(i - 2 * j)	b) fabs(x + y)	c) ceil (x)
d) ceil (x + y)	e) floor (x)	f) floor(x + y)
g) exp(x)	h) log(x)	i) log(exp(x))
j) sqrt(x*x + y*y)	k) sqrt(sin(x) + cos(y))	l) pow(x - y, 3.0)