Informática

Unidad 2G: Subrutinas

Ingeniería en Mecatrónica

Facultad de Ingeniería
Universidad Nacional de Cuyo

UNIVERSIDAD NACIONAL DE CUYO

FACULTAD DE INGENIERIA
en acción continua...

2G – Subrutinas

- Subrutinas
  - Permiten dividir el programa en porciones autocontenidas que pueden ser llamadas múltiples veces y desde múltiples lugares

- Razones para utilizar subrutinas
  - Reutilización de código: una subrutina bien diseñada puede utilizarse en múltiples programas a lo largo del tiempo, reduciendo costos y tiempos de desarrollo. Algunos ejemplos:
    - una función que implemente operaciones sobre matrices
    - un algoritmo de control PID
    - un algoritmo de inteligencia artificial
    - un optimizador
  - División del código
    - Para distribuir el trabajo en equipos de personas
    - Para permitir decidir entre desarrollar o “comprar” distintos componentes
    - Para reducir un problema grande a varios problemas más manejables: enfoque “de lo general a lo particuar”
    - Para aislar componentes mediante interfaces bien definidas y evitar el efecto dominó cuando hay cambios (ripple effect)

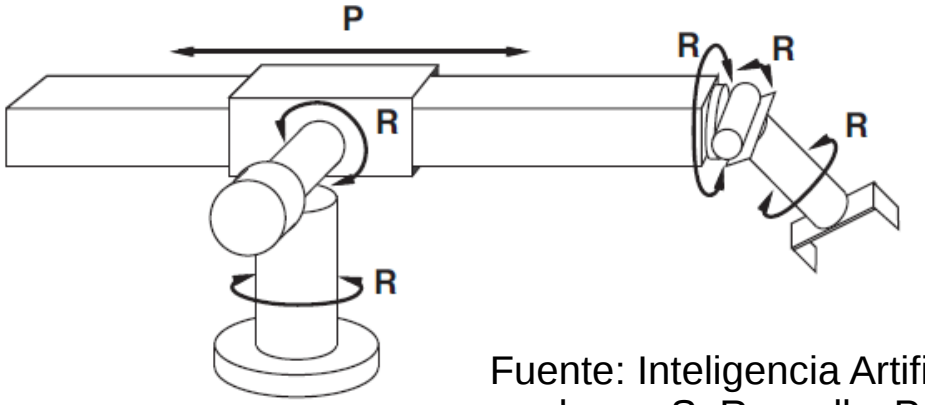
2G – Subrutinas

- Las subrutinas se pueden agrupar en componentes, módulos, subsistemas, para componer un sistema
- Propiedades deseables de las subrutinas y componentes
  - Alta cohesión: las subrutinas que forman parte de un componente están altamente relacionadas (por funcionalidad, por sus estructuras de datos, etc)
  - Bajo acoplamiento: las subrutinas exponen la menor cantidad de parámetros posible al exterior (“esconden” la mayor cantidad de detalles posible en su interior)

- Cómo lograr estas propiedades? ***Encapsulamiento***
  - Qué es? El encapsulamiento consiste en “ocultar” los detalles concretos y las complejidades de la implementación de un algoritmo
  - Para qué? Para que los componentes que utilicen esa funcionalidad no tengan que preocuparse por estos detalles → Simplificación del desarrollo de software

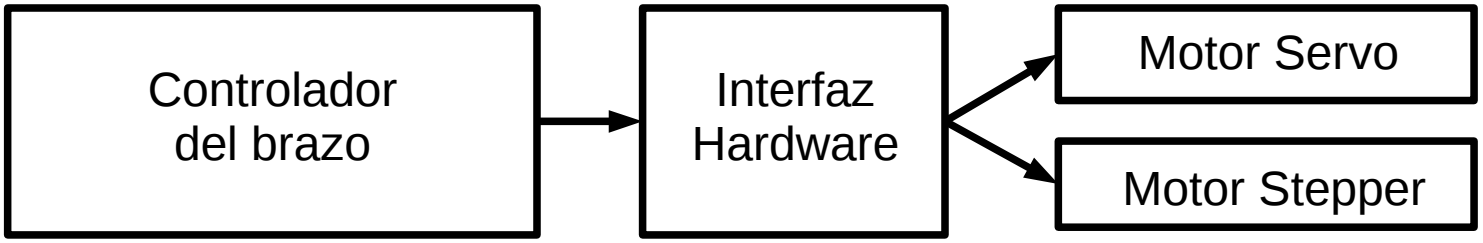
2G – Subrutinas

- Un ejemplo: Manejo de múltiples actuadores similares: steppers y servos



Fuente: Inteligencia Artificial, Un enfoque moderno, S. Russell y P. Norvig, 2004

- Ejercicio: se quiere tener una interfaz uniforme con distintos tipos de motores para mover el brazo



2G – Subrutinas

- Definición de subrutinas
  - Las funciones y procedimientos tienen 2 partes:
    - Cabecera
    - Cuerpo
  - La cabecera, a su vez tiene varias partes: tipo de dato devuelto, nombre de la función, argumentos.
    - Ej:

```
float potencia(float base, int exponente);
```

2G – Subrutinas

- Llamada a funciones
  - Al llamar a una función, se deben pasar los argumentos del tipo de dato correcto, y en el mismo orden definido en la cabecera
    - Ej:

```
float potencia(float base, int exponente);
…
float p = potencia(2.5, 10);
```

  - Los argumentos pueden pasarse
    - por valor: el valor de la variable recibida no cambia al retornar de la función, aun cuando se modifique dentro de la función
    - por referencia: si el valor de la variable se modifica dentro de la función, el cambio se ve reflejado al retornar
  - El pasaje por referencia utiliza *apuntadores* (se verá más adelante)

2G – Subrutinas

- Subrutinas
  - Las subrutinas pueden distribuirse en múltiples archivos fuente (generalmente con extensión **.c**)
  - Para evitar que el compilador de errores al compilar un archivo que utiliza una función que está en otro archivo, se utilizan **archivos de encabezado** (generalmente con extensión **.h**)
  - En cada archivo fuente que utiliza funciones de otro archivo, se coloca la directiva del preprocesador `#include <....h>` correspondiente
  - El IDE (ej: Eclipse) automáticamente compila y enlaza todos los fuentes (en caso de no utilizar eclipse, puede utilizarse el programa ***make*** incluido en GCC para gestionar la compilación de múltiples archivos fuente

2G – Subrutinas

- Subrutinas y Variables
  - Alcance y visibilidad: Variables locales vs. Variables globales
    - Variables locales solo son accesibles en su contexto, Ej:
      - Dentro de la función donde fueron definidas
      - Dentro del bloque donde fueron definidas (puede ser un bucle, una sentencia condicional o simplemente un bloque arbitrario encerrado entre { })
    - Variables globales son visibles desde cualquier parte del programa
  - Ciclo de vida
    - Las variables globales existen (es decir, retienen su espacio de memoria) mientras dura la ejecución del programa
    - Las variables locales se crean y se liberan cuando el programa entra y sale respectivamente de su alcance

- Funcionamiento de llamada a subprogramas

```
int main(){
   sentencia 1
   sentencia 2
   …
   r = funcionA()
   …
   exit(0)
}
```

```
int funcionA(){
   sentencia 1
   sentencia 2
   …
   r = funcionB()
   …
   return r
}
```

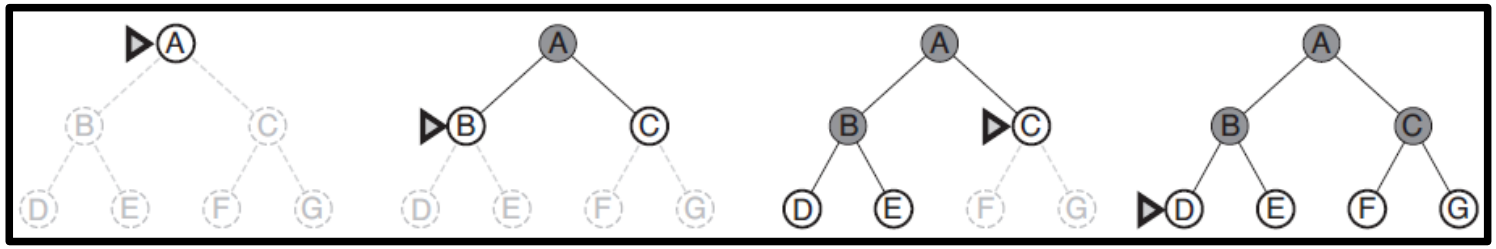
```
int funcionB(){
   sentencia 1
   …
   sentencia n
   return r
}
```

Pila de llamadas

```
main
funcionA
funcionB
```

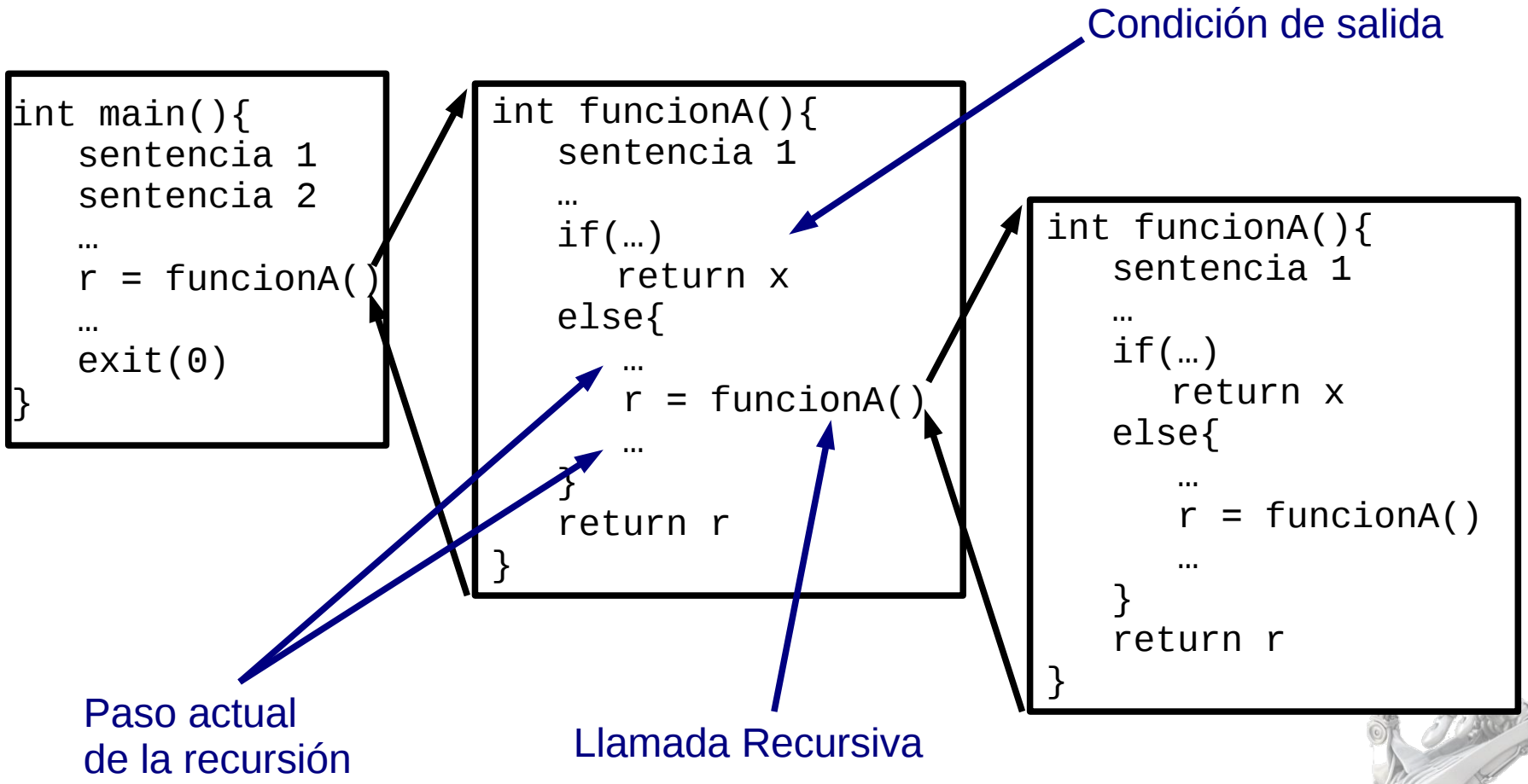
Registro de activación de funciones:
Cada función llamada tiene un espacio de memoria reservado que contiene sus datos privados (valores de los parámetros, puntero de instrucciones, registros, etc.)

- Recursión
  - Un procedimiento es recursivo cuando se llama a sí mismo en algún punto
  - Aplicación: recorrido del árbol



  - Un procedimiento recursivo tiene 3 partes:
    - Una condición de salida
    - El procesamiento del paso recursivo actual
    - La llamada recursiva al paso recursivo siguiente

- Recursión


Condición de salida
int main(){
sentencia 1
sentencia 2
…
r = funcionA()
…
exit(0)
}
int funcionA(){
sentencia 1
…
if(…)
return x
else{
…
r = funcionA()
…
}
return r
}
int funcionA(){
sentencia 1
…
if(…)
return x
else{
…
r = funcionA()
…
}
return r
}
Paso actual
de la recursión
Llamada Recursiva