Programación I

Recursividad

http://proguno.unsl.edu.ar

proguno@unsl.edu.ar

Recursividad

- Técnica de resolución de problemas particulares.
- La definición de un concepto es recursiva si el concepto es definido en términos de sí mismo.
- Ejemplos:
  - Definición de la función factorial.
  - Definición de los números naturales.
  - …

Definiciones recursivas

- En una **definición recursiva**, en general, distinguimos dos partes:
  - Uno o más *casos base* o *elementales*.
  - *Definición recursiva* o *caso general*.

Recursividad en Computación

- Se encuentra presente en:
  - Definiciones recursivas de módulos.
  - Definiciones recursivas de datos.
- La mayor parte de los lenguajes de programación soportan la recursividad.
- Es otra técnica para realizar repeticiones. Aunque no siempre sea la mas adecuada.

Definiciones recursivas de módulos

- Un módulo (función en C) es recursivo cuando:

1. En su cuerpo existe al menos una invocación a sí mismo (caso recursivo o general).

2. Existe uno o varios casos de menor tamaño que pueden resolverse directamente sin necesidad de recursión (caso(s) base(s)).

Ejemplo I: Factorial

Definición matemática de la función factorial

$$!: \mathbb{N}_0 \rightarrow \mathbb{N}$$

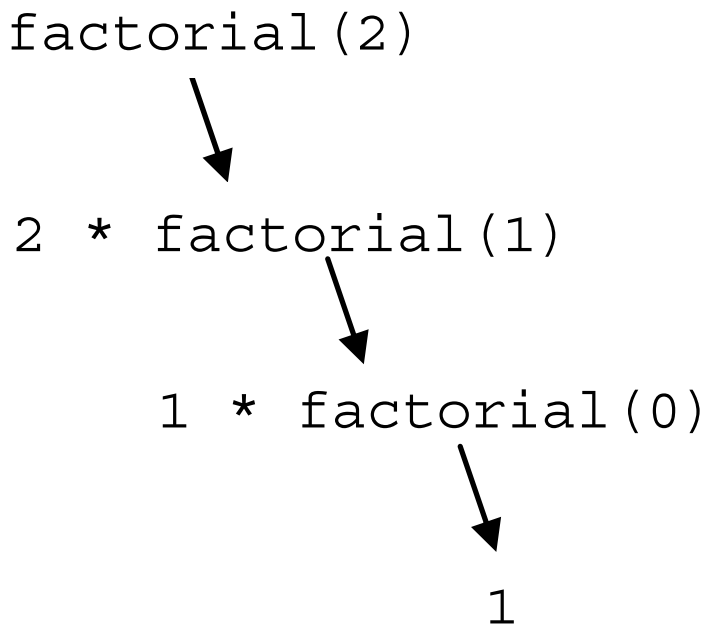
$$n! = \begin{cases} n \times (n\text{-}1)!, & n > 0 \\ 1, & n = 0 \end{cases}$$

Definición de la función factorial en C

```
long factorial(int n){
    if (n == 0)
      return 1;
    else
      return n * factorial(n-1);
}
```

Ejemplo I: Factorial

```
printf("%ld \n", factorial(2));
```

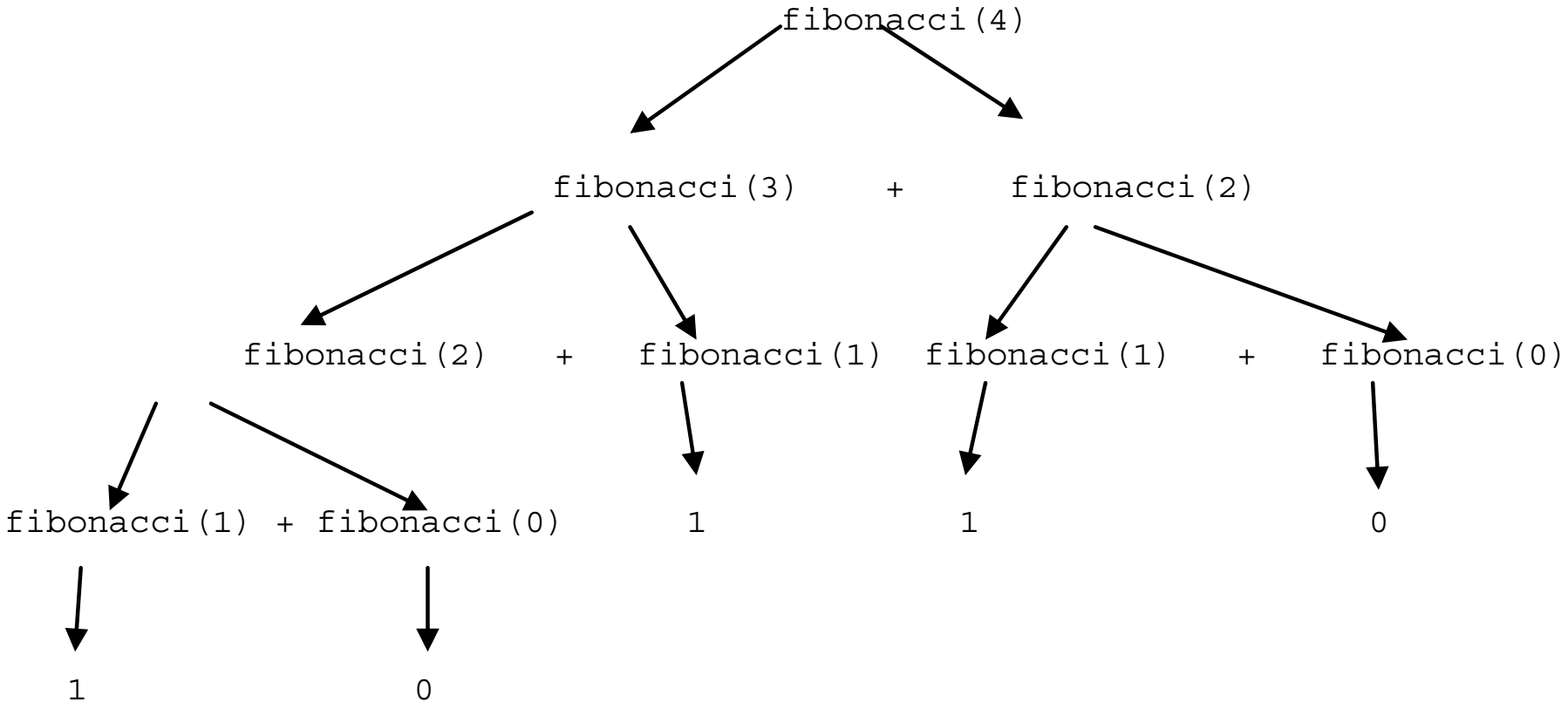


Ejemplo II: Sucesión de Fibonacci

término	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	...
valor	0	1	1	2	3	5	8	13	21	34	55	89	...

- $f_0 = 0$ caso base
- $f_1 = 1$ caso base
- $f_n = f_{n-1} + f_{n-2}$ caso general

Ejemplo II: Sucesión de Fibonacci



Tipos de Recursividad

- Lineal vs. Múltiple
  - Lineal: existe una única invocación recursiva.
  - Múltiple: existe más de una invocación recursiva.
    - Anidada: dentro de una invocación recursiva ocurre como parámetro otra invocación recursiva.

```
int ackerman(int n, int m){
  if (n == 0) return m + 1;
  else if(m == 0) return ackerman(n - 1, 1);
  return ackerman(n - 1, ackerman(n, m - 1));
}
```

Tipos de Recursividad

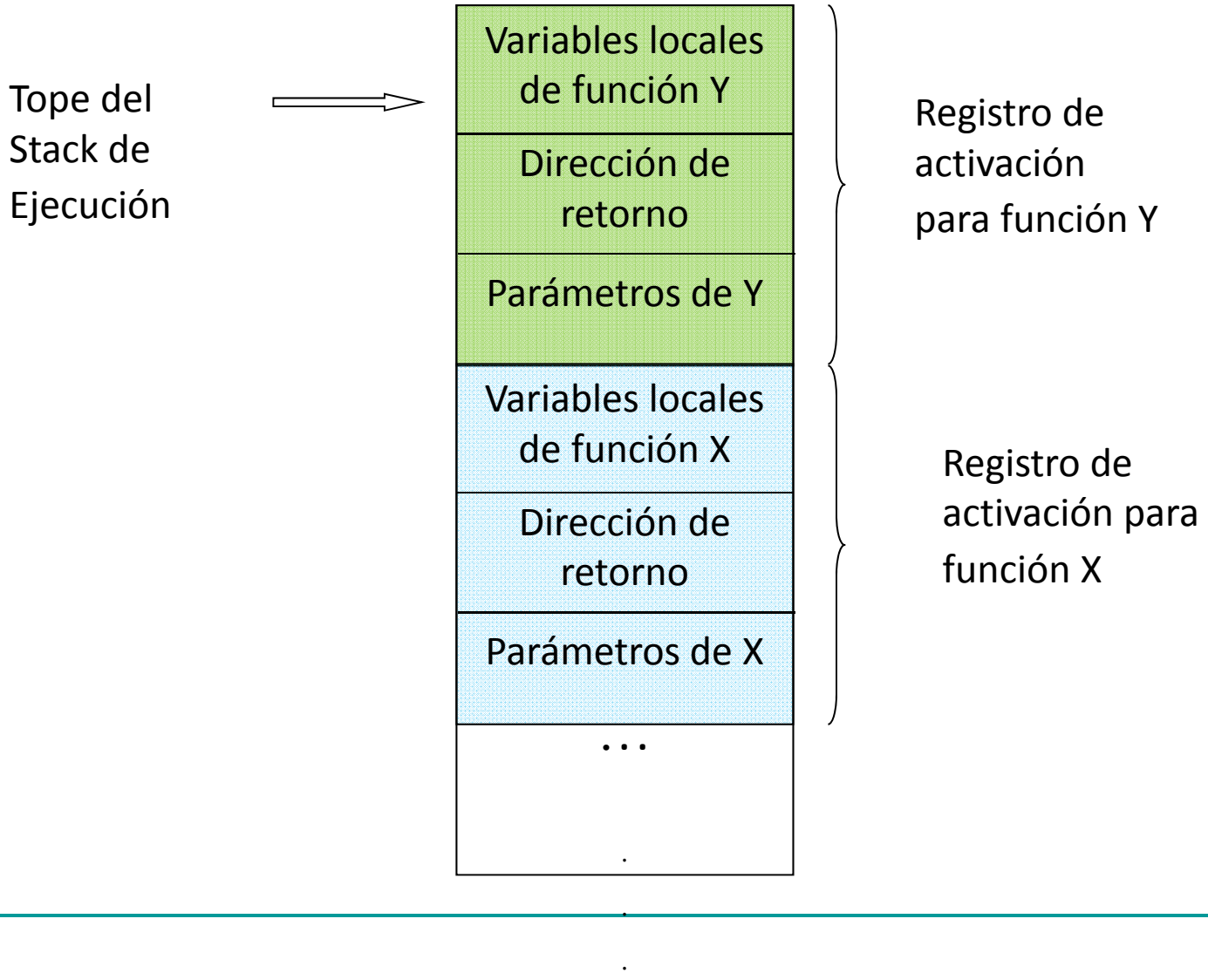
- Directa vs. Indirecta.
  - Directa: el módulo recursivo se llama a sí mismo.
  - Indirecta: se tienen varios módulos que se llaman unos a otros formando un ciclo.
    - función *A* → función *B* → función *A* …
    - función *A* → función *B* → función *C* → función *D* → función *A* …
    - Ejemplo: funciones par e impar:
      - *n* es par si *n* − 1 es impar,
      - *n* es impar si *n* − 1 es par,
      - 0 es par

Stack de Ejecución

- Mantiene la información del estado de ejecución de cada módulo invocado en un programa.

- Guarda los ***registros de activación*** de los módulos invocados.

Stack de Ejecución



Registros de Activación

- Un registro de activación de un módulo almacena el *Ambiente* (variables locales, dirección de retorno, etc.) de un módulo.
- En el tope del stack de ejecución se encuentra el registro de activación del módulo que se está ejecutando.
- Por debajo, los registros de activación de aquellos cuya ejecución aún está pendiente de finalizar.

EJEMPLO

```
1.  int funcion1(int a){
2.    int b;
3.    b = funcion2(a+6);
4.    return a + b;
5.  }
6.  int funcion2 (int c){
7.    return c - 3;
8.  }
9.  int main() {
10.   int b = 3;
11.   printf("%d",b + funcion1(43));
12.   printf("Fin");
13. }
```

Profundidad de la Recursión

- Número máximo de registros de activación en el stack de ejecución para una entrada de datos dada.

Casos particulares

- Inicialización
  - Condición dentro de módulo ☹
  - Usar variables globales ☹
  - Módulos anidados (si lo permite el lenguaje) ☺
  - Antes de la invocación 😐

Casos particulares

- Actualización de valores entre llamadas (p.e. imprimir datos y posiciones)
  - Usar variables globales ☹
  - Usar parámetros extras para pasar valores ☺

Ventajas e Inconvenientes de la Recursividad

- Permite definir un conjunto potencialmente infinito de objetos por medio de una expresión finita. ☺
- Los algoritmos o soluciones recursivas son útiles, particularmente, cuando existe una definición recursiva. ☺
  - Solución compacta.
  - Adaptadas en forma directa a la definición del problema.

Ventajas e Inconvenientes de la Recursividad

- Consumo de tiempo y espacio (creación y destrucción de registros de activación en el stack de ejecución). ☹

- No contribuye a hacer equivalentes las estructuras estática y dinámica del programa .☹
  - Dificulta la depuración del código.
  - Oculta las iteraciones.