

Recursividad

Ing. Armandina Leal Flores

¿Qué es?

- ▶ Propiedad que posee una función que le permite llamarla a sí misma.

Luis Joyanes Aguilar

- ▶ Es una alternativa a la iteración (uso de ciclos).
- ▶ Aunque es menos eficiente en tiempo de computadora en muchas ocasiones permite a los programadores especificar soluciones naturales, sencillas que serían en caso contrario difíciles de resolver.

```
Función 1 (...)  
{  
    //instrucciones  
}  
Función 2 (...)  
{  
    Función 1(...);  
}
```

No hay recursión

Hay recursión

```
Función 1 (...)  
{ .....  
    Función 1 (...)  
    .....  
}
```

Pasos para construir...

- ▶ Definir el problema en términos recursivos
 - Detectar condición de salida (componente Base). Caso en el que debe terminar la recursión.
 - Detectar qué argumentos debe incluir la llamada recursiva para que al cabo de cierta cantidad de llamadas se alcance la condición de salida
- ▶ Escribir la función

Componente.....

- ▶ ¿Qué debe tener la función para llegar a una terminación normal?
 - Un **if** que verifique si se llegó al **caso final**.
 - **Si se llegó al caso final** detener la recursión.
 - **Si no se llegó al caso final** hacer una **llamada recursiva** (continuar la recursión) enviando como parámetro un caso **más pequeño**.

Ejemplo: Recursividad en funciones con return

Factorial del N:

$$N! = (1 \times 2 \times 3 \times 4 \times \dots \times N)$$

Ejemplo:

$$0! = 1$$

$$1! = 1$$

$$2! = 2 \times 1$$

$$3! = 3 \times 2 \times 1$$

...

long factorial (int iN)

{

 if (iN == 0 || iN == 1)

 return 1;

 else return iN * factorial (iN - 1);

}

$$\text{Factorial N} = \begin{cases} 1 & \text{Si } N = 0 \text{ ó } 1 \\ N * (N-1)! & \text{Si } N > 1 \end{cases}$$

```
int main()
{
    cout << factorial (4) << endl;
    return 0;
}
```

Ejemplo: Recursividad en funciones void

- ▶ Desplegar los números de n a cero.

```
void despliegaN (int iN)
{
    if ( iN >= 0 )
    {
        cout << iN << " ";
        despliegaN(iN - 1);
    }
}
```

```
int main()
{
    despliegaN(3);
    cout << endl;
    return 0;
}
```

Ejemplo: Recursividad en arreglos

- ▶ Obtener la suma de los valores almacenados en un arreglo.

```
int obtenerSuma(int iArrA[], int iCant)
{
    if ( iCant == 1 )
        return iArrA[0];
    else return iArrA [iCant-1] + obtenerSuma(iArrA, iCant - 1);
}
```

```
int main()
{
    int iArrA[5] = {1, 2, 3, 4, 5};
    cout << obtenerSuma(iArrA, 5) << endl;
    return 0;
}
```

Ejemplo: Recursividad en arreglos

- ▶ Desplegar el contenido de un arreglo.

```
void despliega(int iArrA[], int iCant)
{
    if ( iCant >= 1 )
    {
        cout << iArrA[iCant-1] << " ";
        despliega( iArrA, iCant - 1 );
    }
}
```

¿Qué pasa si se invierten las instrucciones?

```
void despliega(int iArrA[], int iCant)
{
    if ( iCant >= 1 )
    {
        despliega( iArrA, iCant - 1 );
        cout << iArrA[iCant-1] << " ";
    }
}
```

```
int main()
{
    int iArrA[5] = {1, 2, 3, 4, 5};
    despliega(iArrA, 5);
    return 0;
}
```

Ejemplo: Recursividad en Listas

- ▶ Contar los nodos de una lista

```
int cuenta()
{
    return cuentaNodos(pInicio);
}

int cuentaNodos (Nodo* pActual)
{
    if (pActual == NULL)
        return 0;
    else return 1 + cuentaNodos(pActual -> pSig);
}
```

```
int main()
{
    Lista A;
    -----
    cout << A.cuenta();
    return 0;
}
```

Ejemplo: Recursividad en listas

- ▶ Desplegar el contenido de una lista

```
int despliega ()  
{  
    return despliegaNodos(pInicio);  
}  
  
void despliegaNodos(Nodo* pActual)  
{  
    if (pActual != NULL)  
    {  
        cout << pActual -> iInfo << " ";  
        despliegaNodos (pActual -> pSig);  
    }  
}
```

¿Qué pasa si se invierten las instrucciones?

```
void despliegaNodos(Nodo* pActual)  
{  
    if (pActual != NULL)  
    {  
        despliegaNodos (pActual -> pSig);  
        cout << pActual -> iInfo << " ";  
    }  
}
```

```
int main()  
{  
    Lista A;  
    -----  
    A.despliega();  
    return 0;  
}
```

¿Qué obtiene?

```
int Misterio (int iN)
{
    if ( iN == 0 )
        return 0;
    else return iN + Misterio (iN - 1);
}
```

Suma los números del 0 al n

¿Qué obtiene?

```
int Cifra (int iN, int iD)
{
    if (iD == 0)
        return iN % 10;
    else return Cifra (iN / 10, iD - 1);
}
```

Obtiene el dígito que está en la posición d. Los dígitos se cuentan de derecha a izquierda empezando en 0.

Ejercicios

- ▶ Escribe una función recursiva que despliegue los números de *cero* a *n*.
- ▶ Escribe un método recursivo que genere un string de N asteriscos.
- ▶ Escribe una función recursiva que cuenta la cantidad de pares en una lista lineal
- ▶ Escribe una función recursiva que despliegue los valores impares almacenados en una lista lineal.

Ejercicio

- ▶ ¿Cuál es el error en el siguiente método?

```
int Suma(int iN)
{
    if (iN == 0) return 0;
    return iN + Suma(iN);
}
```

- ▶ ¿Qué hace el siguiente método?

```
int misterio(int iA, int iB)
{
    if (iB == 1) return iA;
    return iA + misterio(iA, iB-1);
}
```