Estructuras de Datos y Algoritmos

Algoritmos Recursivos

Introducción

- Un procedimiento o función recursiva es aquella que se llama a sí misma.
  - La ejecución del proceso recursivo se repite con valores (parámetros) diferentes.
- La recursividad es una alternativa a la iteración muy elegante en la resolución de problemas de naturaleza recursiva. Permite especificar una solución simple y natural para resolver problemas definidos en términos de sí mismos.
  - Ejemplo: Los números naturales
    - 0 es un número Natural
    - El siguiente número de un número natural es otro número Natural

Algoritmos Recursivos

- La recursividad está relacionada con el principio de inducción.
  - Existe un caso base, en el que no existe ninguna llamada recursiva.(Condición o criterio base)
  - Existe un caso general conocido como caso inductivo, en las que se realizan llamadas a versiones de la misma función con parámetros diferentes que conducen al caso base.
- Por lo tanto
  - Hay que incluir por lo menos un caso base, que se resuelva sin necesidad de recursividad.
  - Todas las llamadas recursivas deben llevar hacia el caso base
- El método debe comprobar si se debe realizar una nueva llamada recursiva o si ya se ha alcanzado el caso base.

Algoritmos Recursivos

- El caso base
  - Supone el final de las llamadas recursivas…
  - y la realización de llamadas recursiva que lleva a él lo que evita que se entre en ciclos infinitos.
- Para crear una función recursiva, es necesario tener una definición recursiva del problema.
  - Ejemplo: Suma de los primeros números naturales.

    Caso Base: s(1) = 1;

    Caso general: s(n) = s(n-1) + n

    El problema esta definido en forma recursiva, para conocer la suma de *n* números se debe conocer previamente la suma de los *n-1* números anteriores.

Suma recursiva de los n primeros números naturales

```
int  SumaNat(int n)
{
      int s;
      if (n == 1)
           s = 1; // Caso Base
      else
            s = SumaNat(n-1) + n; // Caso general
       return s;
}
```

Funcionamiento de la recursividad

- Nótese que en una *función recursiva* es necesaria una *condición* para distinguir el caso base del inductivo.
- Para entender cómo funciona la recursividad es necesario tener bien claro que en memoria no existe una sola versión de la función recursiva.
- Cada vez que se invoque la función recursiva se crea una nueva versión de la misma.
- La estructura de todas la versiones es la misma, pero no así los datos que contiene cada una

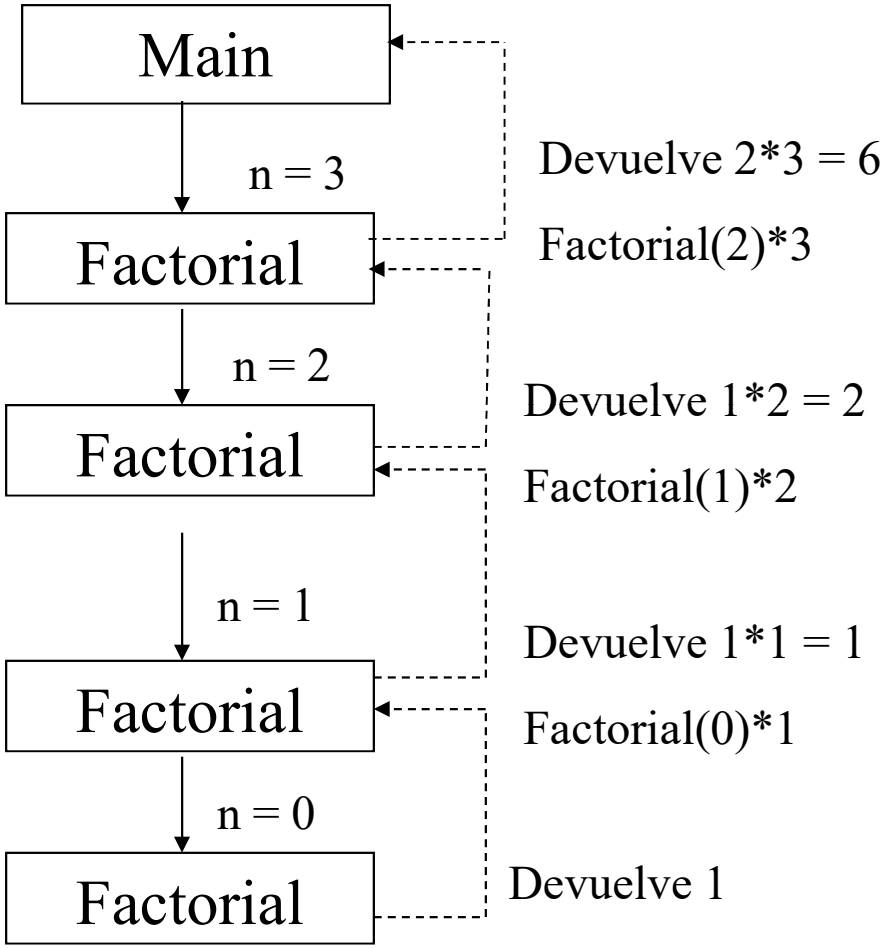
Factorial de un número

Caso base: fact(0) = 1

Caso general: fact(n) = n * fact(n-1)

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

Funcionamiento de la recursividad



IDA VUELTA

Llamada	n	Valor
1ª	3	6
2ª	2	2
3ª	1	1
4ª caso base	0	1

Funcionamiento de la recursividad

- En el anterior ejemplo se mostró cómo las llamadas recursivas se van produciendo hasta alcanzar el caso base.
- En ese punto se acaban las llamadas y empiezan las devoluciones de valores hasta llegar al método *main*.
- Tenemos un movimiento en 2 sentidos

  1º Hacia delante hasta alcanzar el caso base.

  2º Hacia atrás devolviendo los resultados de cada llamada a la función.
- Las llamadas realizadas implican una estructura pila.
- En cada llamada se realiza una copia de la función recursiva (cada llamada implica una nueva copia de las variables de la función). Esto consume memoria.

Correctitud en la recursividad

- ¿Cómo podemos determinar si un Algoritmo Recursivo es o no correcto?
  - Por simple observación es difícil.
  - Es posible alcanzar ese objetivo con la ayuda del principio de inducción.
    - Verificando 1º el caso base, comprobando si devuelve el resultado correcto para el valor más pequeño.
    - Verificando si el algoritmo funciona correctamente para cualquier valor.

Recursividad vs. Iteración

- Características comunes:
  1. Ambas implican repetición
     - La iteración usa explícitamente una estructura de repetición mientras que la recursión logra la repetición mediante llamadas sucesivas a una función.
  2. Ambas requieren de una condición de fin.
     - La iteración termina cuando deja de cumplirse la condición para terminar el ciclo y la recursión cuando se reconoce un caso base.

Recursividad vs. Iteración

3. Ambas se aproximan gradualmente a la terminación.
   - La iteración continua modificando un contador, hasta que éste adquiere un valor que hace que deje de cumplirse la condición del ciclo.
   - La recursividad sigue produciendo versiones más sencillas del problema original hasta llegar al caso base.

Recursividad vs. Iteración

4. Pueden continuar indefinidamente
   - En la iteración ocurre un ciclo infinito, si la condición del ciclo nunca deja de cumplirse.
   - Se tiene una recursión infinita si cada llamada recursiva no simplifica el problema y no se alcanza el caso base o si aún dirigiéndonos al caso base, lo saltamos.

Recursividad vs. Iteración

- Diferencias.
  - La recursividad presenta una desventaja frente a la iteración: la invocación repetida de la función.Cada llamada hace que se cree otra copia de la función esto puede consumir una cantidad excesiva de memoria.
  - La iteración ocurre en la misma función, con lo que se omite el gasto extra de llamadas a la función.
- Toda tarea que pueda realizarse con recursividad puede también realizarse con una solución iterativa.
- Se elige la solución recursiva cuando este enfoque refleja de forma más natural la solución del problema y produce un programa más fácil de entender y depurar.
- Existen problemas cuya solución iterativa no es viable por lo tanto la recursión es una solución.

Aplicaciones de Recursividad

- Los algoritmos recursivos son muy importantes en el diseño de algoritmos:
  - Backtracking (vuelta atrás), búsqueda exhaustiva, usa recursividad para probar todas las soluciones posibles.
  - Divide y vencerás, transforma el problema de tamaño n en problemas más pequeños de tamaño menor que n. De tal modo que en base a problemas unitarios se construye fácilmente una solución del problema completo. Ej. Búsqueda binaria, ordenamiento Quicksort, Torres de Hanoi.

Ejemplo: suma de 2 números usando recursividad

```
int sumaRec(int,int);
int sumaRec(int a, int b)
{
   if(b==0)
              return a;
       else
              if(a==0)
                  return b;
              else
                  return 1+(sumaRec(a,b-1));
}
```

Prueba de escritorio