**Instituto Superior de Formación Técnica Nº 151   
Carrera: Analista en Sistemas  
3 Año. Algoritmos y Estructuras de Datos III.**

|  |  |
| --- | --- |
| **Trabajo Práctico Nº5** | **Unidad 5** |
| **Modalidad:** Semi -Presencial | **Estratégica Didáctica:** Trabajo individual. |
| **Metodología de Desarrollo:** Det. docente | **Metodología de Corrección:** Via Classroom. |
| **Carácter de Trabajo:** Obligatorio – Con Nota | **Fecha Entrega:** A confirmar por el Docente. |

**RECURSION.**

**Marco Teórico:**  
Responder el siguiente cuestionario en función de la bibliografía Obligatoria.  
  
**1**. Que entiende por Recursión.

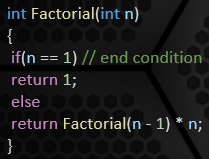
Se llama recursividad a un proceso mediante el que una función se llama a sí misma de forma repetida, hasta que se satisface alguna determinada condición. El proceso se utiliza para computaciones repetidas en las que cada acción se determina mediante un resultado anterior. Se pueden escribir de esta forma muchos problemas iterativos.

2. Que es una función Recursiva y que entiende por Recurrencia , dar ejemplos.

Se llama a una función utilizando un nombre de función y sus parámetros a través de instrucciones. Dada la especificación de entrada-salida de una función, la persona que llama simplemente le hace una llamada. Esta vista de las funciones implica que se invoca, ejecuta y devuelve (con o sin resultados) al lugar donde se llamó en la función de llamada. Cuando una función se llama a sí misma, ya sea directa o indirectamente, se dice que está haciendo una llamada recursiva.

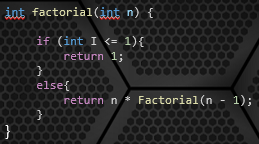
Consideremos un ejemplo de calcular la factorial de un número.   
Factorial es un término matemático. La factorial de un número, digamos n, es igual al producto de todos los enteros de 1 a n. La factorial de n se denota como

n! = 1 x 2 x 3 x ... x n or n! = n x n - 1 x ... x 1

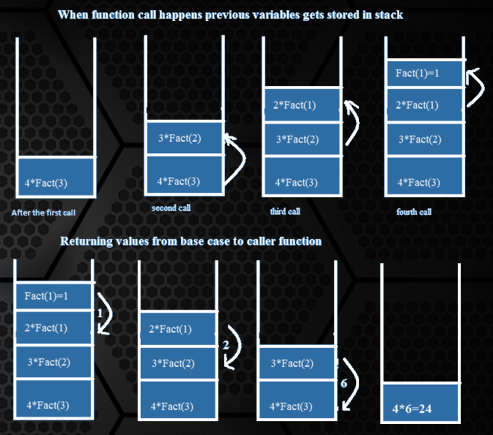
Por ejemplo, ¡10! = 1 x 2 x 3 x 4 x 5 x 6 x 7 x 8 x 9 x 10.   
  
El programa más simple para calcular la factorial de un número es mediante el uso de un bucle con una variable de producto. ¡El algoritmo establece el proceso iterativo de calcular la factorial de n como 10! = 10 x 9 x 8 x ... x 1

La función Factorial() es un ejemplo de una función recursiva. En la segunda instrucción return, la función llama asi misma. Lo importante a recordar al crear una función recursiva es dar una condición final. En el código de programa 4 .1, la recursión se detiene cuando n se convierte en 1. En cada llamada de la función, el valor de n sigue disminuyendo. Sin embargo, cuando el valor alcanza 1, la función termina. Por otro lado, esta función se ejecutará infinitamente si el valor inicial de n es menor que 1, lo que significa que la función no es perfecta. Por lo tanto, la condición n = 1 debe cambiarse a n ≤ 1.

3. Explicar el uso del Stack del sistema operativo en Funciones Recursivas, dar un ejemplo.

La pila es un área especial de memoria donde se almacenan variables temporales. Actúa sobre la base del principio LIFO. Para entender cómo las funciones recursivas usan la pila, analicemos el siguiente código. Los pasos principales se dan en el siguiente código:

Sea n = 3; es decir, calculemos el valor de 3!, que es 3 x 2 x 1 = 6. Cuando la función se llama f o el primer tiempo, n contiene el valor 3, por lo que se ejecuta la instrucción else. La función conoce el valor de n pero no de Factorial(n − 1) por lo que empuja n (valor = 3) dentro de la pila y se llama a sí mismo por segunda vez con el valor 2. Esta vez, la instrucción else se ejecuta de nuevo, y n (valor = 2) se empuja a la pila a medida que la función se llama a sí misma por tercera vez con el valor 1. Ahora, la instrucción if se ejecuta y como n = 1, la función devuelve 1 . Como ahora se conoce el valor de Factorial(1), se revierte a su segunda ejecución extrayendo el último valor 2 de la pila y multiplicándolo por 1. Esta operación da el valor de Factorial(2), por lo que la función vuelve a su primera ejecución extrayendo el siguiente valor 3 de la pila y multiplicándolo por el factorial, dando el valor 6, que finalmente devuelve la función.

1. La función Factorial() se ejecuta tres veces para n = 3, de las cuales se llama a sí misma dos veces. El número de veces que una función se llama a sí misma se conoce como la profundidad recursiva de esa función.
2. 2. Cada vez que la función se llama a sí misma, almacena una o más variables en la pila. Dado que las pilas contienen una cantidad limitada de memoria, las funciones con una alta profundidad recursiva pueden bloquearse debido a la falta de disponibilidad de memoria. Tal situación se conoce como desbordamiento de pila StackOverFlow.
3. Las funciones recursivas generalmente tienen (y de hecho deberían tener) una condición de terminación (o fin). La función Factorial() deja de llamarse a sí misma cuando n = 1. Si esta condición no estuviera presente, la función seguiría llamándose a sí misma con los valores 3, 2, 1, 0, -1, -2, etc. Tal recursión se conoce como recursión sin fin.
4. Todas las funciones recursivas pasan por dos fases distintas. La primera fase, el bobinado, ocurre cuando la función se llama a sí misma y empuja los valores a la pila. La segunda fase, el desenrollamiento, se produce cuando la función saca valores de la pila, generalmente después de la condición final.
5. Explicar las distintas variantes de Recursión.

En C++, los tipos de recursividad se pueden definir en más de una dimensión. En una dimensión, se puede clasificar como **recursividad en tiempo de ejecución y recursión de tiempo de compilación** utilizando metaprogramación de plantillas.

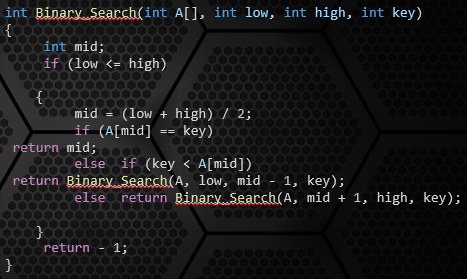
La recursión en **tiempo de ejecución** es la técnica de recursión más común utilizada en C++. Esto se puede implementar **cuando una función de C++ (o función miembro) se llama a sí misma**.

En C++, también podemos hacer recursividad de **tiempo de compilación** con la ayuda de la **metaprogramación de plantillas**. Al crear una instancia de una clase (o estructura) de plantilla en C++, el compilador creará el código de esa clase en tiempo de compilación. Al igual que la recursión en tiempo de ejecución, podemos crear una instancia de la propia clase de plantilla para realizar la recursión. Al igual que la recursión en tiempo de ejecución, también necesitamos la condición de terminación; de lo contrario, la instanciación irá para siempre, al menos teóricamente, pero, por supuesto, limitada a los recursos de la computadora y el compilador. En la metaprogramación de plantillas, podemos especificar la condición de terminación (o condición base) con la ayuda de la especialización de la plantilla o la especialización parcial de la plantilla, dependiendo de la condición de terminación.

La recursión puede tener cualquiera de las siguientes formas:

* Una función se llama a sí misma.
* Una función llama a otra función que a su vez llama a la función de llamada.
* La llamada a la función es parte de la misma instrucción de procesamiento que realiza una llamada a la función recursiva.

Dependiendo de la siguiente caracterización, las funciones recursivas se clasifican como recursiones:

* Directas: cuando una función se llama a sí misma directamente.
* Indirectas: cuando llama a otra función que a su vez la llama.
* Lineales: Se dice que una función recursiva es linealmente recursiva cuando ninguna operación pendiente implica otra llamada recursiva, por ejemplo, la función Factorial(). Esta es la forma más simple de recursión y ocurre cuando una acción tiene una estructura repetitiva simple que consiste en algunos pasos básicos seguidos por la acción nuevamente.
* Árbol: En una función recursiva, si hay otra llamada recursiva en el conjunto de operaciones que se completarán después de que termine la recursión, esto se denomina recursión de árbol. Ejemplos de funciones recursivas de árbol son los algoritmos de ordenación rápida y de combinación, el algoritmo FibSeries, etc.
* Cola: Se dice que una función recursiva es recursiva de cola si no hay operaciones pendientes que realizar a la vuelta de una llamada recursiva. La recursión de cola también se utiliza para devolver el valor de la última llamada recursiva como el valor de la función. La recursividad de cola es ventajosa ya que la cantidad de información que debe almacenarse durante el cálculo es independiente del número de llamadas recursivas. La función Factorial() en el Código de Programa es un ejemplo de una función recursiva no tail. La función Binary\_Search() en el código de programa es un ejemplo de una función recursiva de cola.
* **Recursión binaria** Una función recursiva binaria se llama a sí misma dos veces. El cálculo de números de Fibonacci, la ordenación rápida y la ordenación de combinación son ejemplos de recursión binaria
* **Recursión n-ary y permutaciones** La forma más general de recursión es la recursión n-ary, donde n no es una constante sino algún parámetro de una función. Las funciones de este tipo son útiles para generar objetos combinatorios como las permutaciones.

1. Describir las diferencias entre Iteración y Recursión.

La recursión es un enfoque de arriba hacia abajo de la resolución de problemas. Divide el problema en pedazos o selecciona un paso clave, posponiendo el resto. Por otro lado, la iteración es más un enfoque de abajo hacia arriba.

Aunque con muchos méritos, los algoritmos recursivos tienen sus limitaciones:  
1. Muchos lenguajes de programación no soportan recursión;

2. Aunque las funciones matemáticas se pueden implementar fácilmente utilizando la recursión, siempre es a costa de tiempo de ejecución adicional y espacio de memoria.

3. Se puede llamar a una función recursiva desde dentro o fuera de sí misma, y para garantizar un funcionamiento adecuado, debe guardar las direcciones de retorno en algún orden para que el retorno a la ubicación adecuada produzca el resultado deseado cuando se realice el retorno a una instrucción de llamada.

Aunque el método iterativo tiene varios méritos, también tiene sus propias limitaciones:  
1. El código iterativo no es legible y, por lo tanto, no es fácil de entender.   
2. En las técnicas iterativas, el bucle de las declaraciones es necesario y necesita una lógica compleja.   
3. Las iteraciones pueden dar lugar a un código largo.

**Marco Practico:**

1. Escribir dos funciones mutualmente recursivas par(n) e impar(n) que determinen la paridad del numero natural dado, conociendo solo que:
   1. 1 es impar.
   2. Si un número es impar, su antecesor es par; y viceversa.

Lic. Oemig José Luis.