UNIVERSIDAD AUTÓNOMA CHAPINGO

DPTO. DE PREPARATORIA AGRÍCOLA

ÁREA DE FÍSICA

“RECURSIÓN”

Guillermo Becerra Córdova

E-mail: gllrmbecerra@yahoo.com

OBJETIVOS:

Este trabajo tiene por objetivo mostrar las características principales de la RECURSIÓN. La recursión es una de las herramientas de programación mas poderosas, siendo un proceso alterno que es utilizado para cálculos repetitivos en los que cada acción se determina en función de un resultado anterior. La mayoría de los problemas pueden resolverse de una manera directa usando métodos no recursivos; sin embargo, otros pueden resolverse de una manera más lógica y elegante mediante la recursión. En este trabajo desarrollaremos técnicas para encontrar soluciones recursivas a algunos problemas que también pueden resolverse por métodos iterativos. La idea de esto es mostrar la lógica de la programación recursiva a través de problemas sencillos. Finalmente, mencionaremos las ventajas y desventajas que implica la programación recursiva.

REFERENTE TEÓRICO:

Cuando preguntamos por el significado de cierta palabra, es muy común que dentro de la respuesta figure alguna relacionada con la misma que se quiere definir. La mayoría de los diccionarios definen ciertas palabras en términos más simples de la misma. Por ejemplo, «descripción» se define en términos más simples: acción y efecto de describir. En consecuencia, para saber el significado de «descripción», se requiere ahora conocer el significado de «describir». Si en esta definición figura de nuevo una palabra similar a las anteriores, el proceso se repetiría ya que la definición que se busca no estaría del todo completa, porque no se ha definido en otros términos. Afortunadamente, los diccionarios definen los verbos en términos diferentes a los que se quiere definir, finalizando con ello este proceso. De igual forma, en Matemáticas y en Computación existen definiciones que se pueden establecer en términos de casos más simples del mismo. A tales definiciones se les llama recursivas. Por ejemplo, nos hemos encontrado con la necesidad de conocer el valor de la suma de los primeros números naturales. Esta suma es muy empleada cuando queremos conocer el monto de la cantidad que se juntará en una rifa, donde el valor del boleto corresponde a la cantidad que tiene que pagar el concursante. Si son 100 boletos y cada número representa la cantidad en pesos que tiene que pagar, el monto total sería de $ 5050.00. Se puede obtener el resultado sumando 100 al resultado de la suma de 1+2+3+...99: o sumando 100+99 al resultado de la suma de 1 +2+3+...98; etc. Esta forma de sumar se puede escribir de manera mas compacta: suma(100)=100 + suma(99) = suma(100)=100+99+ suma(98), etc. En general, suma(n)=n + suma(n-1). Observe que la definición anterior está establecida en términos más simples a lo que se quiere definir. Diremos en este caso que es una definición recursiva. Existen muchos ejemplos donde este criterio es empleado: el producto de números naturales, la suma o producto de cualquier conjunto de números, la serie de Fibonacci, el calculo de determinantes, la serie de Ackerman, las Torres de Hanoi, etc. Sin embargo, debemos mencionar, que al igual que las definiciones del diccionario, en la recursión debe existir una condición de termino de proceso; es decir, una definición que esté expresada en otros términos o en forma no recursiva. Por ejemplo, en la suma de los primeros números naturales, debe existir un término donde la definición esté expresada en forma diferente para establecer el fin del proceso. Si definimos que suma(1) sea igual a 1, entonces estaríamos en posibilidades de conocer el valor de suma(2)=2 + suma(1), de igual forma es posible conocer suma(3), suma(4), suma(5) hasta suma(100). La definición recursiva nos dice que antes de evaluar suma(100) debe evaluarse suma(99); antes de evaluar suma(99), debe evaluarse suma(98); etc. Por tanto al repetir el proceso se tiene que: suma(100)=100+suma(99)=100+99+suma(98)=…=100+…+suma(1)

Cada caso se reduce a uno mas simple hasta que llega a suma(1), la cual se define de manera directa como 1. Recuerde que suma(1) está definida en forma directa, no como la suma de otro numero. Esto hace posible regresar la suma, regresando la suma calculada para evaluar el resultado de la suma anterior.

Como mencionamos anteriormente, puede implementarse un algoritmo recursivo para calcular

la suma de los primeros 100 números naturales. Un requisito importante para que sea correcto un algoritmo recursivo es que no genere una secuencia infinita de llamadas a sí mismo. Cualquier algoritmo que genere tal secuencia, no terminará. Una función recursiva debe definirse alguna vez en términos que no impliquen la definición misma. Debe existir una condición de salida o de paro de proceso de la secuencia de llamadas recursivas. Para la suma, esta condición corresponde a suma(1)=1. Sin esta salida no recursiva, no puede calcularse ninguna función recursiva. Cualquier caso de definición recursiva o invocación de un algoritmo recursivo tiene que reducirse a la larga a alguna manipulación de uno o más casos simples no recursivos.

A continuación mostraremos un programa recursivo en lenguaje C que sirve para calcular la suma de los primeros números naturales:

```
int suma(int n)
{
 if(n==1)
 return(1)
 else
 return(n+suma(n-1)
}
```

Como se puede observar, este lenguaje permite al programador definir subrutinas y funciones que se llamen a sí mismas (n+suma(n-l)), las cuales también se llaman recursivas. Observe que la función recursiva se define en términos que no impliquen a la función al menos una vez return(1). La palabra return sirve para indicar que la función debe regresar un valor, que en este caso es entero, como es expresado con las letras iniciales int de la palabra integer. Para calcular la suma de los primeros 100 números naturales, simplemente se hace una llamada a la función pasando como argumento el número 100, es decir: suma(100). Cuando es llamada la función suma(100), el parámetro n se iguala con 100. Como n no es 1, se llama a la función por segunda vez con un argumento de 99. Por lo tanto la función suma actúa de nuevo y como n no es igual a 1, la función suma es llamada por tercera vez, ahora con un argumento de 98, y así sucesivamente hasta que el argumento de la función se iguala con 1. En esta situación ya no se llama de nuevo la función recursivamente, sino que toma el valor de 1 porque así se ha definido de forma no recursiva. En este punto regresan en orden inverso las funciones que fueron llamadas recursivamente. La primera de ellas, después de la definición no recursiva, es suma(2)=2+suma(1), la segunda es suma(3)=3+suma(2), etc., hasta llegar a suma(100) que es el valor que se quería calcular. Observe que cada vez que regresa una rutina recursiva, lo hace al punto que sigue de manera inmediata a aquél desde la cual se llamó. Así, la llamada recursiva a suma(1), sirve para calcular suma(2)=2+suma(1), suma(2) sirve para calcular suma(3)=3+suma(2), etc. Es importante aclarar que la función no regresará ningún valor si no se han evaluado todos los parámetros de la misma. Por ejemplo, la función suma(100) no regresará algo hasta que se haya evaluado suma(99), tampoco suma(99) regresará algún valor si no se ha evaluado suma(98), etc. Las funciones regresarán un valor cuando se evalúe suma(1). Cada vez que se llama de manera recursiva una función, se asigna un nuevo conjunto de variables y parámetros que son liberados al regresar la función. De esta forma se va asignando espacio de memoria cada vez que sea llamada una función recursiva y esa memoria es liberada al regreso de la misma. Es por ello que es utilizada mucha memoria en un proceso recursivo. Si la cantidad de memoria utilizada sobrepasa la cantidad de memoria de la máquina, el programa se detendrá. Es por ello que un proceso recursivo no es tan eficiente, pero sí elegante.

Si en vez de calcular la suma de números naturales, se calculara el producto (conocido como factorial), el algoritmo correspondiente sería similar al anterior, con la única diferencia de sustituir el signo de suma por el signo de producto. El algoritmo resultante es el siguiente:

```
int factorial(int n)
```

```
{
 if(n==1)
 return(1)
 else
 return(n*factorial(n-1)
}
```

Aquí la definición no recursiva corresponde al caso en que n sea igual a 1, asignándole un valor de 1, es decir, la definición no recursiva establece que factorial(1)=1. Para valores de n mayores a 1, el proceso de llamadas recursivas es muy similar al de la suma. Por ejemplo, para conocer factorial(10), la función llamará a factorial(9), que a su vez llamará a factorial(8), etc., hasta que el valor de n sea igual a 1, En ese momento se evalúa el factorial(1) y comienza el proceso inverso para evaluar factorial(2), factorial(3), etc., hasta llegar al valor establecido originalmente.

Existen funciones recursivas en las que la funci6n se refiere dos veces a sí misma. Tal es el caso de la secuencia de Fibonacci que es la secuencia de enteros:

0, 1, 1, 2, 3, S, 8, 13, 21, 34,…

Cada elemento de esta sucesi6n es la suma de los dos precedentes. En este caso se tienen que establecer dos definiciones no recursivas: Fibonacci(0)=0 y Fibonacci(1)=1. Entonces puede definirse la secuencia de Fibonacci mediante la definición recursiva:

Fibonacci(n) = n si n = 0 o n = 1

De lo contrario:

Fibonacci(n) = Fibonacci(n-2) + Fibonacci(n-1)

Por ejemplo, para calcular Fibonacci(8), puede aplicarse la definición recursiva para obtener lo siguiente:

Fibonacci(8)= Fibonacci(6)+ Fibonacci(7)=Fibonacci(4)+ Fibonacci(5)+ Fibonacci(5)+ Fibonacci(6)=Fibonacci(2)+ Fibonacci(3)+ Fibonacci(3)+ Fibonacci(4)+ Fibonacci(3)+ Fibonacci(4)+ Fibonacci(4)+ Fibonacci(5)=Fibonacci(0)+ Fibonacci(1)+ Fibonacci(1)+ Fibonacci(2)+ Fibonacci(1)+ Fibonacci(2)+ Fibonacci(2)+ Fibonacci(3)+ Fibonacci(1)+ Fibonacci(2)+ Fibonacci(2)+ Fibonacci(3)+ Fibonacci(2)+ Fibonacci(3)+ Fibonacci(3)+ Fibonacci(4)=0+1+1+ Fibonacci(2)+1+ Fibonacci(2)+ Fibonacci(2)+ Fibonacci(3)+1+ Fibonacci(2)+ Fibonacci(2)+ Fibonacci(3)+ Fibonacci(2)+ Fibonacci(3)+ Fibonacci(3)+ Fibonacci(4)=2+ Fibonacci(2)+1+ Fibonacci(2)+ Fibonacci(2)+ Fibonacci(3)+1+ Fibonacci(2)+ Fibonacci(2)+ Fibonacci(3)+ Fibonacci(2)+ Fibonacci(3)+ Fibonacci(3)+ Fibonacci(4)=2+ Fibonacci(0)+ Fibonacci(1)+1+ Fibonacci(0)+ Fibonacci(1)+ Fibonacci(0)+ Fibonacci(1)+ Fibonacci(1)+ Fibonacci(2)+1+ Fibonacci(0)+ Fibonacci(1)+ Fibonacci(0)+ Fibonacci(1)+ Fibonacci(1)+ Fibonacci(2)+ Fibonacci(0)+ Fibonacci(1)+ Fibonacci(1)+ Fibonacci(2)+ Fibonacci(1)+ Fibonacci(2)+ Fibonacci(2)+ Fibonacci(3)=2+0+1+1+0+1+0+1+1+ Fibonacci(0)+ Fibonacci(1)+1+0+1+0+1+1+ Fibonacci(0)+ Fibonacci(1)+0+1+1+ Fibonacci(0)+ Fibonacci(1)+1+ Fibonacci(0)+ Fibonacci(1)+ Fibonacci(0)+ Fibonacci(1)+ Fibonacci(1)+ Fibonacci(2)=7+0+1+4+0+1+2+0+1+1+0+1+0+1+1+ Fibonacci(0)+ Fibonacci(1)=20+0+1=21

Un programa en lenguaje C para establecer la secuencia de Fibonacci requiere de dos definiciones no recursivas y cada vez se llama dos veces las funciones recursivas. A continuación presentamos el programa que reproduce tal secuencia:

```
int fibonacci(n)
{
 if(n==0 || n==1)
 return(n)
 else
 return(fibonacci(n-2)+Fibonacci(n-1))
}
```

Observe en el código que si el valor de n es igual a 1 o 0, la función Fibonacci regresa un valor igual al valor de n, que corresponde a las definiciones no recursivas. Para el caso de la definición recursiva, se tiene que la función llama dos veces a la misma función, pero con diferentes argumentos: uno en una unidad y el otro en dos unidades menor a la del argumento original. Ambas llamadas a la función son sumadas como se muestra en el código. Este programa ilustra cómo puede llamarse a sí misma varias veces una función recursiva con argumentos diferentes.

RESULTADOS Y DISCUSIÓN:

En la figura 1 se muestra la ventana principal del sistema que se elaboró como resultado del proyecto. En ella aparece el nombre del tema a tratar, el lugar donde se elaboró, los autores y finalmente dos opciones: Salir y Continuar. Si el usuario no desea continuar, solo debe hacer clic en el botón Salir para detener el programa.

Figura 1

En caso contrario, al hacer clic en el botón Continuar, aparecerá la ventana que se muestra en la figura 2. En ella se muestran las opciones: Suma, Producto, Potencia, Raíz Cuadrada, Fibonacci y Torres de Hanoi. Estas opciones muestran algunos de los problemas que pueden ser resueltos con la Recursión. El usuario podrá activar cualquiera de ellas con solo hacer clic en la opción correspondiente.

Figura 2

Una de las opciones que presenta el sistema es el problema conocido como Torres de Hanoi, que consiste en tres postes A, B y C. En el poste A se coloca cualquier número de discos de diámetro diferente de tal manera que un disco de diámetro mayor siempre quede debajo de un disco de diámetro menor. Observe la figura3.

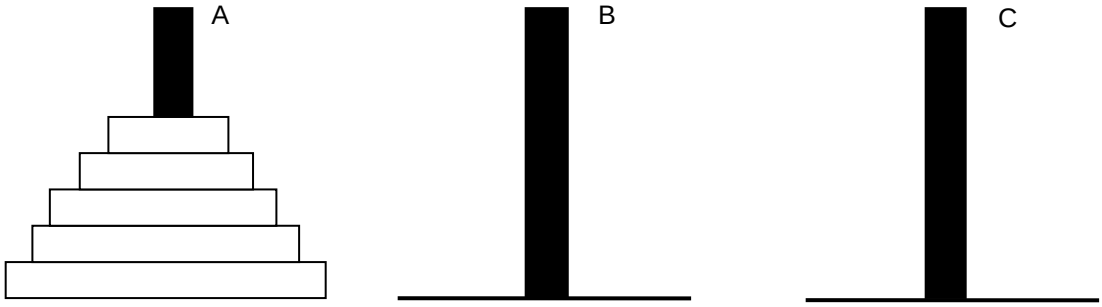

Figura 3

El objetivo es mover los discos al poste C usando el B como auxiliar. Solo puede moverse el disco superior de cualquier poste a otro poste y un disco mayor jamás podrá quedar sobre uno menor. La recursión presenta una manera elegante de solucionar el problema planteado. La siguiente lista muestra los pasos que se deben seguir para el caso de tres discos.

1. MOVER DISCO 1 desde A hasta C
2. MOVER DISCO 2 desde A hasta B
3. MOVER DISCO 1 desde C hasta B
4. MOVER DISCO 3 desde A hasta C
5. MOVER DISCO 1 desde B hasta A
6. MOVER DISCO 2 desde B hasta C
7. MOVER DISCO 1 desde A hasta C

CONCLUSIONES:

- En muchas ocasiones la solución recursiva es la vía más lógica y natural de resolver un problema, ya que la solución recursiva se desprende de manera directa de la definición de recursividad.
- Las ideas vertidas aquí pueden ser aplicadas a problemas más complejos.
- En general una versi6n no recursiva de un programa se ejecutará con mayor eficacia en cuanto a tiempo y espacio que una recursiva. Sin embargo, la solución recursiva puede ser la forma más simple de resolver un problema.

BIBLIOGRAFÍA:

- Tenenbaum, Aaron M;Langsam, Yedldyah; Augemtetn, Mouse A. Estrocturas de Datos en C. Prentice Hall. 1993.
- Garcfa de Sola, Juan F; Garcerm He~ez, Vlcente. Lenguaje C y Estructuras de Datos. Aplicaciones Generales y de Gesti6n. Mc Graw Hill. 1992.
- Gottfried, Byron S. Programaci6n en C. Mc Graw Hill. 1994. Joyanes Aguilar, Lu~. Turbo Pascal. Mc Graw Hill. 1993.
- CebaUos, Francisco Javier. Enciclopedia del Lenguaje C. Addison- Wesley Iberoamericana. 1991.
- Turbo Pascal con Aplicaciones Hennereld, JuUen. Gru1X> Editoriallberoamtrica. 1989.