Presentación:

Tarea 2: Investigación de recursión

Nombre: Farfán de León José Osvaldo.

Código: 214796622.

Carrera: Ingeniería en computación.

Materia: Estructura de datos.

Profesor: Julio Esteban Valdez López.

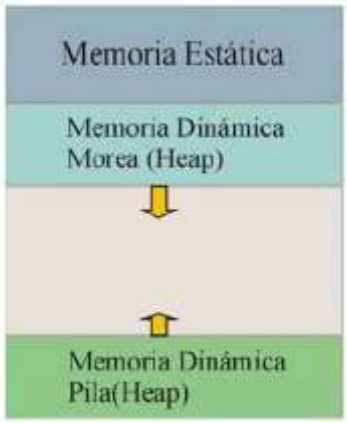
Fecha de entrega:16/10/2020.

16 de noviembre de 2020

Recursión:

Una función recursiva es una función que se llama a sí misma. Esto es, dentro del cuerpo de la función se incluyen llamadas a la propia función. Esta estrategia es una alternativa al uso de bucles.

Cuando un programa llama a una función que llama a otra, la cual llama a otra y así sucesivamente, las variables y valores de los parámetros de cada llamada a cada función se guardan en la pila o stack, junto con la dirección de la siguiente línea de código a ejecutar una vez finalizada la ejecución de la función invocada. Esta pila va creciendo a medida que se llama a más funciones y decrece cuando cada función termina. Si una función se llama a si misma recursivamente un número muy grande de veces existe el riesgo de que se agote la memoria de la pila, causando la terminación brusca del programa.



A pesar de todos estos inconvenientes, en muchas circunstancias el uso de la recursividad permite a los programadores especificar soluciones naturales y sencillas que sin emplear esta técnica serían mucho más complejas de resolver. Esto convierte a la recursión en una potente herramienta de la programación.

Tipos de recursión:
Como regla básica, para que un problema pueda resolverse utilizando recursividad, el problema debe poder definirse recursivamente y, segundo, el problema debe incluir una condición de terminación porque, en otro caso, la ejecución continuaría indefinidamente. Cuando la condición de terminación es cierta la función no vuelve a llamarse a si misma. Pueden distinguirse distintos tipos de llamada recursivas dependiendo del número de funciones involucradas y de cómo se genera el valor final. A continuación, veremos cuáles son.

Recursión lineal:
En la recursión lineal cada llamada recursiva genera, como mucho, otra llamada recursiva. Se pueden distinguir dos tipos de recursión lineal atendiendo a cómo se genera resultado.

Recursión lineal no final:
En la recursión lineal no final el resultado de la llamada recursiva se combina en una expresión para dar lugar al resultado de la función que llama. El ejemplo típico de recursión lineal no final es cálculo del factorial de un número (n! = n * (n-1) * ...* 2 * 1). Dado que el factorial de un número n es igual al producto de n por el

factorial de n-1, lo más natural es efectuar una implementación recursiva de la función factorial. Veamos una implementación de este cálculo:

```
/*
*ejemplo7_1.c
*/
#include <stdio.h>

int factorial(int numero){
  if (numero > 1) return (numero*factorial(numero-1));
  else return(1);
}

int main (){
  int n;
  printf("Introduce el número: ");
  scanf("%d",&n);
  printf("El factorial es %d", factorial(n));
}
```

Recursión lineal final

En la recursión lineal final el resultado que es devuelto es el resultado de ejecución de la última llamada recursiva. Un ejemplo de este cálculo es el máximo común divisor, que puede hallarse a partir de la fórmula:

$$mcd(n,m)=\begin{cases} n & n=m \\ mcd(n-m,m) & n>m \\ mcd(n,m-n) & n<m \end{cases}$$

```
/*
*ejemplo7_2.c
*/
#include <stdio.h>

long mcd(long,long);

main(int argc, char *argv[]){
   long a= 4454,b= 143052;
   printf("El m.c.d. de %ld y %ld es %ld\n",a,b,mcd(a,b));
}
```

```
long mcd(long a, long b){
   if (a==b) return a;
   else if (a<b) return mcd(a,b-a);
   else  return mcd(a-b,b);
}
```

Recursión mutua

Implica más de una función que se llaman mutuamente. Un ejemplo es el determinar si un número es par o impar mediante dos funciones:

```
/*
*ejemplo7_4.c
*/
#include <stdio.h>

long fibonacci (int);

int main(){
  int n= 30;
  if (par(n))
    printf("El número es par");
    else
    printf("El número es impar");
}

int par(int n){
  if (n==0) return 1;
  else return (impar(n-1));
}

int impar(int n){
  if (n==0) return 0;
```

```
  else return(par(n-1));
}
```

Veamos un ejemplo de ejecución de este programa:

```
par(5) -> return(impar(4))     return(0)  (no es par)
impar(4) -> return(par(3))     return(0)
par(3) -> return(impar(2))     return(0)
impar(2) -> return(par(1))     return(0)
par(1) -> return(impar(0))     return(0)
```

impar(0) -> return(0)

Bibliografía.

Bravo Lastra, M., & Tarazona Cierto, Y. (2017). Aplicar recursividad. Recuperado 15 de octubre de 2020, de SENATI website: https://sites.google.com/site/portafoliosenati/fundamentos-de-programacion/13-aplicar-recursividad

http://biolab.uspceu.com/aotero/recursos/docencia/TEMA%207.pdf

16 de noviembre de 2020