Escuela Politécnica del Ejército

RECURSIVIDAD

Cuaderno de Estructuras de Datos

En el siguiente documento podrán encontrar temas como:

1. Algoritmos recursivos.
2. Casos en los que debe evitarse el uso de la recursividad .
3. Recursividad directa e indirecta.
   1. Metodos para la resolucion de problemas que usan recursividad .
   2. Divide y venceras.
   3. Backtracking(retroceso).
4. Ejemplos de Recursividad.

2023

Tercero   
Curso :14675

**RECURSIVIDAD.**

La recursividad (recursión) es aquella propiedad que posee una función por lo cual dicha función puede llamarse así misma. La recursión es una herramienta poderosa e importante en resolución de problemas y en programación. Se puede utilizar la recursividad como una alternativa a la iteración. Una solución recursiva es normalmente menos eficiente en términos de tiempo de computadora que una solución iterativa debido a las operaciones auxiliares que llevan consigo las llamadas suplementarias a las funciones; sin embargo, en muchas circunstancias el uso de la recursión permite a los programadores especificar soluciones naturales, sencillas, que serían, en caso contrario, difíciles de resolver. (Luis Aguilar, 2005)

1. **Algoritmos recursivos.**

Una función recursiva es aquella que se llama a si misma o bien directamente o bien a través de otra función. Una función que tiene sentencias entre las que se encuentra al menos una que llama a la propia función se dice que es recursiva. La definición de factorial de un número natural n se define como para todo número mayor que 0, y Una programación recursiva de la función factorial puede consultarse en el Ejercicio 1. En el diseño de un algoritmo. (Luis Aguilar, 2005)

* El instrumento necesario para expresar los programas recursivamente es la función.
* Una función tiene recursividad directa cuando se llama a sí misma dentro de su propia definición.
* Una función tiene recursividad indirecta si llama a otra función que, a su vez, contiene una referencia directa o indirecta a la primera.
* El cuerpo de una función recursiva debe tener una o varias instrucciones selectivas donde establece para los casos generales las soluciones generales, y para los casos triviales las soluciones triviales.
* Cada llamada recursiva debe disminuir el tamaño del ejemplar de entrada de tal manera que se acerque a las soluciones triviales.
* Cada vez que se llama a una función recursiva todos los valores de los parámetros formales y variables locales son almacenados en una pila.
* Cuando termine de ejecutarse la función recursiva retorna al nivel inmediatamente anterior. En la pila se recuperan los valores tanto de los parámetros como de las variables locales, y se continúa con la ejecución de la siguiente instrucción a la llamada recursiva.

1. **Casos en los que debe evitarse el uso de la recursividad .**
2. **Recursividad Directa e Indirecta**

**Directa:**

La recursividad puede considerarse de cabeza, cola o intermedia según que las operaciones se hagan después, antes o antes y después de la llamada recursiva. Por otra parte, también existe la recursividad múltiple, con diversas llamadas recursivas y la anidada.

**Ejemplo**

**Recursividad de Cabeza:** En este tipo de recursividad, las operaciones se realizan antes de la llamada recursiva.

#include <iostream>

void recursionDeCabeza(int n) {

if (n > 0) {

// Operaciones antes de la llamada recursiva

std::cout << n << " ";

// Llamada recursiva

recursionDeCabeza(n - 1);

}

}

int main() {

std::cout << "Recursividad de Cabeza: ";

recursionDeCabeza(5);

return 0;

}

En este ejemplo, se imprimen los números en orden descendente antes de realizar la llamada recursiva.

**Indirecta:**

La recursividad indirecta se produce cuando una función llama a otra, que eventualmente termina llamando de nuevo a la primera función.

#include <iostream>

void funcionB(int n); // Declaración anticipada

void funcionA(int n) {

if (n > 0) {

std::cout << n << " ";

funcionB(n - 1);

}

}

void funcionB(int n) {

if (n > 1) {

std::cout << n << " ";

funcionA(n / 2);

}

}

int main() {

std::cout << "Recursividad Indirecta: ";

funcionA(10);

return 0;

}

En este ejemplo, funcionA llama a funcionB, y luego funcionB llama de nuevo a funcionA, creando así un ciclo de llamadas recursivas indirectas.

1. **Metodos para la resolucion de problemas que usan recursividad .**
2. **Divide y venceras.**
3. **Backtracking(retroceso).**

**Ejercicio 1.** La función suma impares emplea la recursividad para hallar la suma de los n primeros números enteros impares**.**

#include <iostream>

using namespace std;

double sumaimpares(int);

int main () {

int n;

cout << "Escriba un numero entero positivo: \n";

cin >> n;

cout << "Suma de los " << n << " primeros impares: " << sumaimpares(n) << endl;

return 0;

}

double sumaimpares(int n) {

if (n <= 1)

return 1.0;

return ((2 \* n - 1) + sumaimpares(n - 1));

}

# **Bibliografía**

Luis Aguilar, M. F. (2005). *Estructuras de datos en C.* Madrid: McGraw-Hill.