Recursividad

¿Qué es recursividad?

- Que puede repetirse.
- La recursividad es un concepto fundamental en matemáticas y en computación.
- Es una alternativa diferente para implementar estructuras de repetición (ciclos). Los módulos se hacen llamadas recursivas.
- Se puede usar en toda situación en la cual la solución pueda ser expresada como una secuencia de movimientos, pasos o transformaciones gobernadas por un conjunto de reglas no ambiguas.

Función recursiva

Se compone de:

- **Caso base:** una solución simple para un caso particular (puede haber más de un caso base). Es importante determinar un caso base, es decir un punto en el cual existe una condición por la cual no se requiera volver a llamar a la misma función.
- **Caso recursivo:** una solución que involucra volver a utilizar la función original, con parámetros que se acercan más al caso base. Los pasos que sigue el caso recursivo son los siguientes:
  1. La función se llama a sí misma
  2. El problema se resuelve, tratando el mismo problema pero de tamaño menor
  3. La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará.

Ejemplo: Cálculo del factorial

Calcule el factorial (!) de un entero no negativo.

0!	=1	=1	=1
1!	=1 * 0!	=1* 1	=1
2!	=2 * 1!	=2 * 1	=2
3!	=3 * 2!	=3 * 2 * 1	=6
4!	=4 * 3!	=4 * 3 * 2 * 1	=24
5!	=5 * 4!	=5 * 4 * 3 * 2 * 1	=120
N!	**N * (N – 1)!**		

El factorial de un número es la multiplicación de cada número desde 1 hasta ese número, entonces es muy sencillo crear un ciclo de 1 hasta el número pedido para hacer el cálculo.

Factorial (sin recursividad)

```
#include <iostream>
using namespace std;

int main(int argc, char *argv[]) {
    int i=0, num=0, resultado=1;
    cout<<"Calculo del factorial."<<endl;
    cin>>num;
    for(i=1;i<=num;i++){
        resultado=resultado*i;
    }
    cout<<"El factorial de "<<num<<" es: "<<resultado;
    return 0;
}
```

Uso de la estructura repetitiva FOR

Factorial (con recursividad)

Para el cálculo de **N!**

1 Si N=0 CASO BASE

N*(N-1)! Si N>0 CASO RECURSIVO

Factorial (con recursividad)

```
int calcularFactorial(int num){
    if(num==0){
        return 1;
    }else{
        return num*calcularFactorial(num-1);
    }
}
```

Caso Base

Se llama así misma

Caso Recursivo

Factorial (con recursividad)

```
#include <iostream>
using namespace std;

int calcularFactorial(int);

int main(int argc, char *argv[]) {
   int num=0;
   cout<<"Calculo del factorial."<<endl;
   cin>>num;
   cout<<"El factorial de "<<num<<" es: "<<calcularFactorial(num);
   return 0;
}

int calcularFactorial(int num){
  …
}
```

¿Por qué usar recursividad?

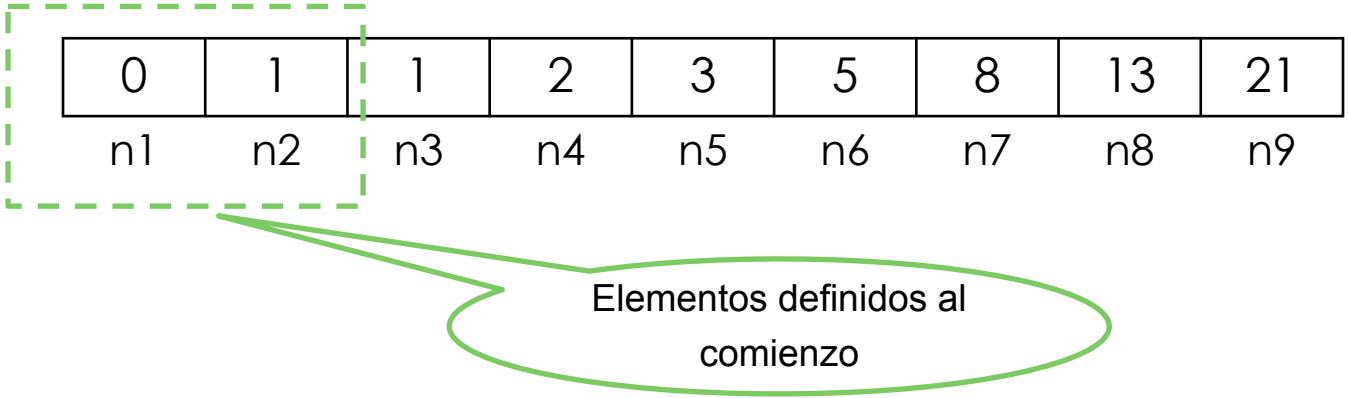
- Son más cercanos a la descripción matemática.
- Permiten simplificar el código.
- Generalmente más fáciles de analizar.
- Se adaptan mejor a las estructuras de datos recursivas.
- Los algoritmos recursivos ofrecen soluciones estructuradas, modulares y elegantemente simples.

¿Cuándo no usar recursividad?

- Cuando las funciones usen arreglos largos.
- Cuando las funciones cambian de manera impredecible.
- Cuando las iteraciones sean la mejor opción.

Ejemplo: Cálculo de la Serie de Fibonacci

Los valores son:



Ejemplo: Cálculo de la Serie de Fibonacci

Comienza con un 0, luego un 1 y a partir de ahí cada término de la serie suma los 2 anteriores. **Fórmula recursiva:**

$$fib(n) = fib(n - 1) + fib(n - 2)$$

$$fib(3) = fib(3-1) + fib(3-2)$$

$$= fib(2) + fib(1)$$

$$=1+0$$

Ejemplo: Cálculo de la Serie de Fibonacci

```
…

#define n1 0
#define n2 1

int calcularFibonacci(int);

int main(int argc, char *argv[]) {
   …
}

int calcularFibonacci(int num){
   if(num==1){
      return n1;
   }else if(num==2){
      return n2;
   }else{
      return calcularFibonacci(num-1)+calcularFibonacci(num-2);
   }
}
```

Caso Base

Se llama así misma

Caso Recursivo

Ejemplo: Cálculo de la Serie de Fibonacci

```
#include <iostream>
using namespace std;

#define n1 0
#define n2 1

int calcularFibonacci(int);

int main(int argc, char *argv[]) {
   int i=0, num=0;
   cout<<"Ingrese cuantos numeros de la serie Fibonacci desea ver:"<<endl;
   cin>>num;
   for(i=1;i<=num;i++){
       cout<<calcularFibonacci(i)<<"-";
   }
   return 0;
}
int calcularFibonacci(int num){
   if(num==1){
       return n1;
   }else if(num==2){
       return n2;
   }else{
       return calcularFibonacci(num-1)+calcularFibonacci(num-2);
   }
}
```

Recursión vs iteración

Repetición

- Iteración: ciclo explícito (se expresa claramente)
- Recursión: repetidas invocaciones a una función

Terminación

- Iteración: el ciclo termina o la condición del ciclo falla
- Recursión: se reconoce el o los casos bases

En ambos casos podemos tener ciclos infinitos

Guía de Ejercicios

Autor

Realizar un programa que solicite el ingresar de un número y mostrar su equivalente en binario usando una función recursiva. Por ejemplo:
SISTEMA DECIMAL SISTEMA BINARIO
2 10
3 11
4 100
5 101

```
int pasar_binario(int numero){
	//si numero es mayor que 1, puedo seguir dividiendo por 2
	if(numero>1)
		pasar_binario(numero/2);
	//Muestro el resto de la division de numero dividido 2.
	cout<<numero%2; //% obtiene el resto de una division
	return 0;
}
```

Determinar la suma de los primeros "n" números naturales.
Por ejemplo: Ingresa 3. Entonces debería mostrar 1+2+3=6.

```
int sumar(int num){
	//caso bases
	if(num==0)
		return 0;
	if(num==1)
		return 1;
	else
		//caso recursivo
		return num+sumar(num-1);
}
```

Determinar la potencia de un número con funciones recursivas.
Ejemplo: 5 a la 3 =125

```
int calcular_potencia(int base,int exp){
	if(exp==0)
		return 1;
	if(exp==1)
		return base;
	else
		return base*calcular_potencia(base,exp-1);
}
```

Determinar el producto de dos números de manera recursiva. Por ejemplo:
3 * 4 = 1
3+(3*3) = 12
3+(3+(3*2)) = 12
3+(3+(3+(3*1)))=12

```
int calcular_producto(int n1,int n2){
	string cadena="";
	if(n1==0||n2==0){
		return 0;
	}else if(n1==1){
		return n2;
	}else if(n2==1){
		return n1;
	}else{
		cout<<"\n"<<n1<<"+("<<n1<<"*"<<n2-1<<")="<<n1*n2<<"\n"; //solo para mostrar
		return n1+calcular_producto(n1,n2-1);
	}
}
```

Calcular el Máximo Común Divisor de dos números. Ejemplos:
MCD(15,20)=5
MCD(6,9)=3
MCD(16,21)=1

```
int calcular_MCD(int n1, int n2){
      if(n2==0) //caso base
            return n1;
      else    //caso recursivo
            return calcular_MCD(n2,n1%n2);
}
```

Ingresar un número y mostrar el número de forma invertida ejemplo 123 - 321. Nota: Se está utilizando el sistema decimal, es por ello que utilizamos el 10.

```
void invertir(int num){
	//se utiliza el 10 porque estamos en el sistema decimal.
	// % se obtiene el resto de la division.
	cout<<num%10;

	if (num>10) //caso recursivo

		invertir(num/10);
}
```

Ingresar una palabra y determinar si es palindroma o no. (Un palíndromo es una palabra, número o frase que se lee igual hacia adelante que hacia atrás. Si se trata de un número, se llama capicúa. Ejemplo: Neuquen)

```
int palindroma(char palabra[],int ini, int fin){
	if(palabra[ini] == palabra[fin]){
		palindroma(palabra, ini+1, fin-1);
		return 1;
	}
	else{
		return 0;
	}
}
```

Obtener los números primos del 1 al 100, es decir que debe imprimir los siguientes números:

1-2-3-5-7-11-13-17-19-23-29-31-37-41-43-47-53-59-61-67-71-73-79-83-89-97

```
bool esPrimo (int num, int divisor){
	if(num/2<divisor){
		return true;
	}else if (num%divisor==0){
		return false;
	}else{
		return esPrimo (num,divisor+1);
	}
}
```

Cargar un arreglo con 10 valores y ordenarlo con el método recursivo quick sort.

ver pdf de quiksort