Funciones Recursivas

Es llamada función recursiva a aquella que se llama a si misma, en general tiene un comportamiento cíclico, podriamos decir tambien que se comporta como un bucle while, do-while o for .

A medida que hemos estado avanzando, nos hemos dado cuenta que:

1. Los Bucles while y do-while son las formas mas básicas de estructuras cíclicas.

2. El bucle for, podria considerarse como una evolución de los bucles while y do-while. Ya que nos brinda una versión mas compacta e intuitiva que sus predecesores por tener tres campos en su estructura; inicialización,condición y actualización.

Ahora, agregaremos un punto mas a esta lista de descubrimientos:

3. Las funciones recursivas son consideradas una evolución de todos los bucles anteriores, debido a que nos permitira el uso óptimo del numero de variables a usar, resolver ciclos mas complejos e implementar programas de forma mas organizada y compacta.
Además se puede afirmar que cualquier bucle puede ser expresado como una función recursiva y cualquier función recursiva puede ser expresada como un bucle.

While
do-while

for

Funciones
Recursivas

ESTRUCTURA GENERAL :

```
tipo NOMBRE_FUNCIÓN (tipo argumento1,tipo argumento2) {

        if (Condición_parada) {
                return numero;
        }
        else {
                Sentencias que modifican los argumentos;
                return NOMBRE_FUNCIÓN (argumento1,argumento2);
        }

}
```

EJEMPLO PRÁCTICO – FUNCIONES RECURSIVAS

Resolveremos el ejercicio de la clase anterior (Sucesion de Fibonnaci) mediante un bucle for y luego mediante funciones Recursivas.

Mediante Bucle for :

```
#include<stdio.h>
```

Mediante Funciones recursivas :

```
#include<stdio.h>
```

ARRAYS

La necesidad de trabajar con una gran cantidad de datos, nos hace preguntarnos, ¿cómo los guardaremos? (¿Debemos declarar tantas variables como numeros?), para este tipo de problemática, C nos facilita una solución muy efectiva : Los Arrays!!

Un Array es una estructura que guarda datos del mismo tipo, usted cada vez que piense en un array, relacionelo con los vectores muy usados en Fisica (hablamos en este caso de arrays unidimensionales) o con las matrices de Algebra lineal (hablamos en este caso de arrays multidimensionales).

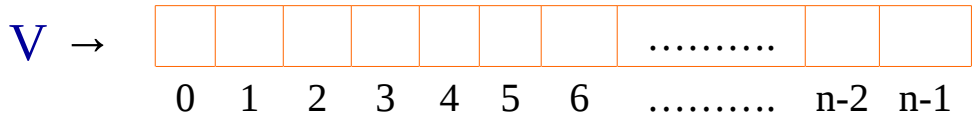
Arrays unidimensionales

CASO ESTÁTICO:
Para hacerlos existir, escribimos:
tipo Nombre_array[n];

Ejemplo: int V[n];

CASO DINÁMICO:
tipo *Nombre_array;
Nombre_array =(tipo *) malloc (n * sizeof(tipo));

Ejemplo: int *V;
V= malloc (n *sizeof(int));



Para acceder a contenido: V[n]={4,2,7,44,6,9,33,75,…..,44} ó V[n-1]=13

En la memoria:

Dirección	Contenido	Nombre
1000		V[1]
1004		V[2]
1008		V[3]
1012		V[4]
1016		V[5]
1020		V[6]
1024		V[7]
…..	…..	…..
20X		V[n]

Observaciones:

- Aunque parezca una estructura compleja,trabaje con V[i] como si fuera una variable de tipo entero.
- Es de buenas practicas, cuando se trabaja de forma dinámica que se libere la memoria usada al finalizar el programa, usando la función free. Ejemplo: free(V);

Ejercicio aplicativo:

1) Ingresar 10 números por teclado, mostralos, ordenarlos y mostrarlos nuevamente; finalmente muestre la suma, resta, multiplicación y división de todos.

Sugerencia: Hagalo primero de forma estática, para luego hacerlo de forma dinámica y que el programa funcione con la cantidad de numeros que se desee al ejecutar el programa.

LIBRO DE REFERENCIA

Introducción a la programación con C.

by Andrés Marzal Varó; Isabel Gracia Luengo

link de acceso a libro:

https://ia800808.us.archive.org/0/items/2010IntroduccionALaProgramacionConC/2010_introduccion-a-la-programacion-con-c.pdf