Funciones Recursivas

Es llamada funcion recursiva a aquella que se llama a si misma, en general tiene un comportamiento ciclico, 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 ciclicas.

2. El bucle for, podria considerarse como una evolucion de los bucles while y do-while. Ya que nos brinda una version mas compacta e intuitiva que sus predecesores por tener tres campos en su estructura; inicializacion,condicion y actualizacion.

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 optimo del numero de variables a usar, resolverciclos mas complejos e implementar programas de forma mas organizada y compacta.
Ademas se puede afirmar que cualquier bucle puede ser expresado como una funcion recursiva y cualquier funcion recursiva puede ser expresada como un bucle.

While
do-while

for

Funciones
Recursivas

ESTRUCTURA GENERAL :

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

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

}
```

EJEMPLO PRACTICO – 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, ¿Como los guardaremos? (¿Debemos declarar tantas variables como numeros?), para este tipo de problemática, C nos facilita una solucion muy rapida : 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).

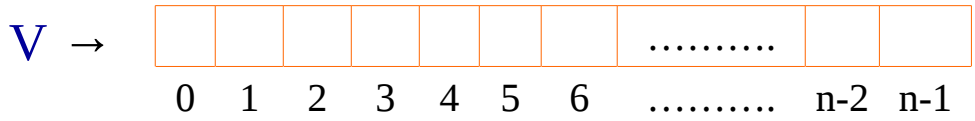
Arrays unidimensionales

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 dinamica que se libere la memoria usada al finalizar el programa, usando la funcion free. Ejemplo: free(V);

Ejercicio aplicativo:

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

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

Funciones

#include<stdio.h>

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

    Sentencia1;
    Sentencia2;
    Sentencia3;
    return resultado;

}



void main(){

    tipo variable1;
    tipo variable2;
    tipo almacena;
    Sentencias;
// llamada a funcion:
    almacena = NOMBRE (variable1,variable2);
    Sentencias;

}
```

PASO POR VALOR :

```
tipo NOMBRE (tipo argumento1,tipo argumento2){
    Sentencia1;
    Sentencia2;
```

```
    Sentencia3;
    // Si en un momento dado las variables argumento1 y argumento2 cambian
    // este cambio no afecta directamente a las variables variable1 y variable2 que
    // se encuentran en la funcion main().
    ……
    return resultado;
}
```

PASO POR REFERENCIA :

```
tipo NOMBRE (tipo *argumento1,tipo *argumento2){
    Sentencia1;
    Sentencia2;
    Sentencia3;
    // En este caso si podemos modificar directamente los valores de las variables
    // variable1 y variable2 que se encuentran en la funcion main().
    // Solo es necesario modificar *argumento1 o *argumento2.
    return resultado;
}

void main(){

    tipo variable1;
    tipo variable2;
    tipo almacena;
    Sentencias;
// llamada a funcion por valor:
    almacena = NOMBRE (variable1,variable2);
// llamada a funcion por referencia:
    almacena = NOMBRE (&variable1,&variable2);
    Sentencias;
}
```

Ejercicios aplicativos:

1) El valor de la función e^x puede aproximarse con el desarrollo de Taylor:

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Diseña una función que aproxime el valor de e^x usando n términos del desarrollo de Taylor, siendo n un número entero positivo. (Puedes usar, si te conviene, la función de exponenciación del último ejercicio para calcular los distintos valores de x^i, aunque hay formas más eficientes de calcular $x/1!$, $x^2/2!$, $x^3/3!$, ..., ¿sabes cómo? Plantéate cómo generar un término de la forma $x^i/i!$ a partir de un término de la forma $x^{i-1}/(i-1)!$.)

2) Imprimie los n primeros terminos de la sucesion de fibonacci. Resolverlo con una funcion recursiva.

Solución :

LIBRO DE REFERENCIA

Introducción a la programación con C.

by Andrés Marzal Varó; Isabel Gracia Luengo

Revisar Estructuras de Control Iterativas (pags **60-68)** y Funciones (pags **152-176)**.

link de acceso a libro:

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