

La programación estructurada es una técnica para escribir programas orientados a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras básicas: secuencia, selección (if y switch) e iteración (bucles for y while).

**Set up de codeblocks y compilador.**

Un **entorno de desarrollo integrado** o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

**Code :: Blocks** es un IDE multiplataforma gratuito y de código abierto que admite múltiples compiladores, incluidos GCC, Clang y Visual C ++.

Se desarrolla en C ++ usando wxWidgets como el kit de herramientas GUI.

[Descargar codeblocks](http://www.codeblocks.org/downloads)

**¿Qué es un compilador?**  
Es el encargado de traducir el lenguaje de programación que utilizamos al lenguaje maquina (lenguaje binario), es decir ceros y unos.  
**¿Qué hace un compilador?**

Como lo dice su nombre se encarga de compilar, en este caso compila nuestros programas transmitiéndolos a la máquina para que nuestro programa funcione.  
**¿Qué es codeblocks?**  
Es un IDE openSource que nos servirá para programar en C y compilar.  
**¿Qué es openSource?**  
Es software que está disponible para su modificación.

# Variables y tipos de datos (Asignación y clasificación)

# Al realizar nuestros programas tendremos que hacer uso de las siguientes fases:

# 

# Un tipo de dato es una clasificación que el programador le da a la información almacenada para avisarle al compilador cómo va a ser interpretada.

# 

# Una variable es un espacio reservado en memoria, definido por un tipo de dato y un nombre asignado, en el cual se puede guardar un valor y se puede modificar.

# 

# 

# *Una variable está declarada cuando indicamos solo su tipo de dato, mientras que cuando decimos que una variable esta inicializada es porque ya contiene y se le ha asignado un valor.*

# Entrada y salida de datos

# En un programa, los datos de entrada son los que la computadora va a procesar. Los datos de salida son datos derivados de los procesos internos que va a realizar nuestro programa.

# 

# Operadores aritméticos

Los operadores los utilizamos para realizar operaciones aritméticas en nuestras funciones. Algunos operadores son:

* Suma: +
* Resta: -
* Multiplicación: \*
* División: /
* Módulo: %

La operación **módulo** obtiene el resto de la división (***residuo***) entre dos números.

# 

# Operadores de asignación.

# Utilizamos estos operadores de asignación para simplificar la manera en que asignamos valores a nuestras variables. Lo que se puede ver en las dos columnas son maneras diferentes en la que podemos escribir estas asignaciones.

# 

# Operadores de incremento y decremento.

# Estos también son operadores que nos ayudarán a simplificar una operación.

# 

# Operadores relacionales.

# 

# Operadores lógicos.

# 

# Condicional if e if - else

Las **condicionales** son instrucciones que evalúan resultados booleanos (true or false), generalmente usados para alterar el flujo de un programa.

Una **sentencia condicional** es una instrucción o grupo de instrucciones que se pueden ejecutar o no en función del valor de una condición.

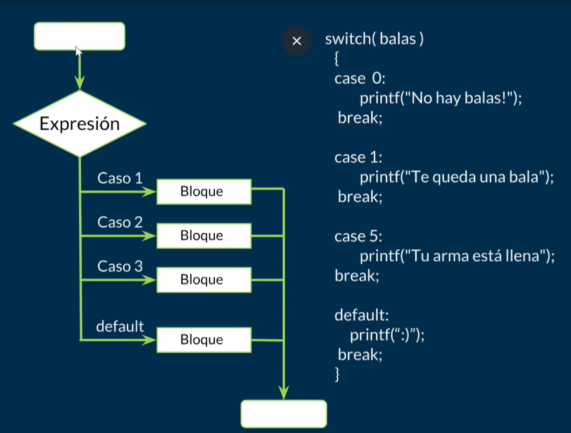
Ejemplo:

# 

# Switch

**Switch** es una estructura de control para agilizar el flujo del programa en opciones múltiples.

Se tiene una condición que se puede cumplir dependiendo de diferentes casos, cada caso nos lleva a un bloque de cumplimiento de la condición y ese bloque nos lleva a continuar el programa.



# ¿Qué es un iterador? - Ciclo While

Un loop es una estructura iterativa que permite repetir un bloque de instrucciones. Esta repetición es controlada por una condición booleana.

Un **iterador** es utilizado por un algoritmo para recorrer los elementos almacenados en un contenedor. Dado que los distintos algoritmos necesitan recorrer los contenedores de diversas maneras para realizar diversas operaciones, y los contenedores deben ser accedidos de formas distintas, existen diferentes tipos de iteradores.

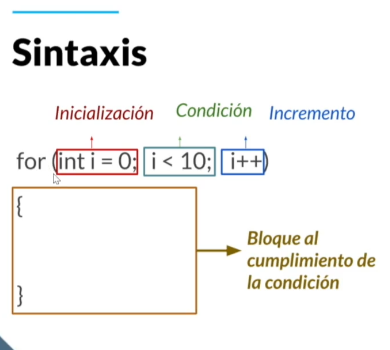
**While**  
Es una estructura de control en la que la repetición se realizará tantas veces como se indique mientras se cumpla una condición.

# 

# Ciclo For

El ciclo **for** es una estructura de control que nos permite repetir un bloque de comandos un número de veces específico.

Este ciclo se compone de tres partes:  
1- Inicialización: Se inicializa una variable (generalmente ““i””) al asignarle el valor 0.  
2- Condición: Se indica la condición necesaria para que termine el ciclo.  
3- Incremento: También puede ser decremento; este va a indicar los pasos en los que se moverá el ciclo, este número siempre debe ser entero, para ir de uno en uno se utiliza el i++.



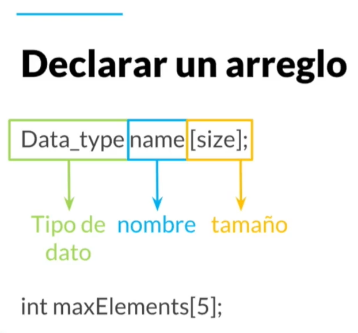
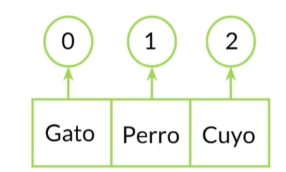
# Ciclo Do-While

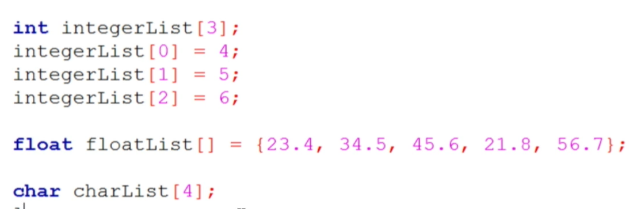
El **Do - While** es una estructura de control donde la condición de continuación del ciclo se prueba al final del mismo.  
Funciona de manera similar a la estructura while, la diferencia es que esta evalúa **al final**.

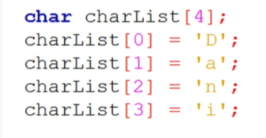
# 

# Arreglos unidimensionales (Arrays).

Un **arreglo** es una serie de elementos, del mismo tipo de dato y son almacenados de manera consecutiva. En programación, la primera posición de los arreglos empieza en “0”.

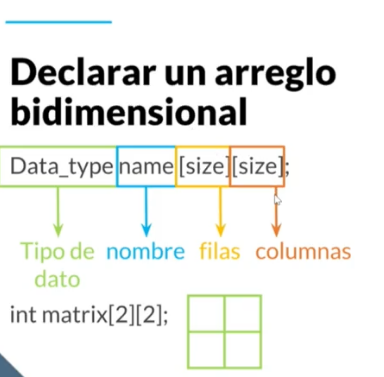
 

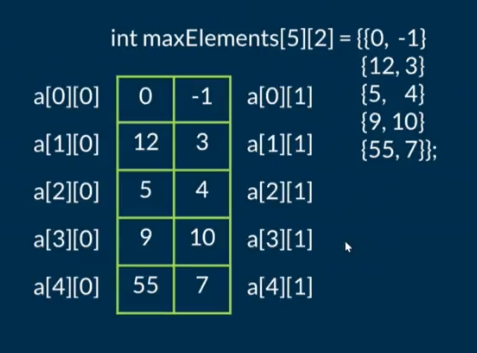
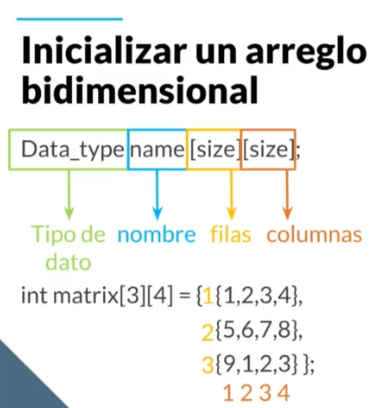




# Arreglos bidimensionales (Matrices).

Los **arreglos bidimensionales** son también llamados tablas o **matrices**.  
Tiene dos índices: el primero indica el número de **fila** y el segundo el número de **columna** en que se encuentra el elemento.

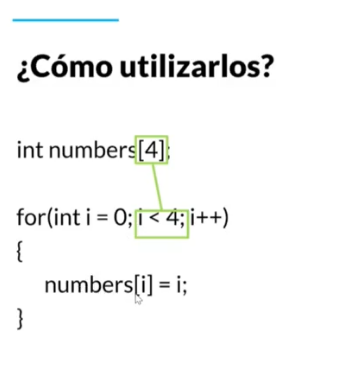




**Arreglos e iteradores unidimensionales**

**¿Por qué utilizar iteradores en arreglos unidimensionales?**

* Para manipular todos los elementos de un arreglo podemos utilizar una estructura repetitiva. La más usual es el ciclo for.
* Cuando se desea imprimir el contenido del arreglo
* Cuando se suman todos los elementos
* También cuando se va a inicializar el arreglo.



# Arreglos e iteradores bidimensionales.

Para poder utilizar el iterador ***for*** junto con un arreglo bidimensional es necesario entender el concepto de un ***for anidado***. Esto es un for dentro de un for, la sintaxis es la siguiente:

# 

# En esta estructura al iniciar el for, se recorre vuelta por vuelta ambos fors, iniciando por la primera vuelta del primer for y continuando con todas las vueltas del segundo for. Siguiendo con la siguiente vuelta del primer for y continuando con todas las vueltas del segundo for, y así sucesivamente.

# Esto es muy útil para recorrer arreglos bidimensionales.

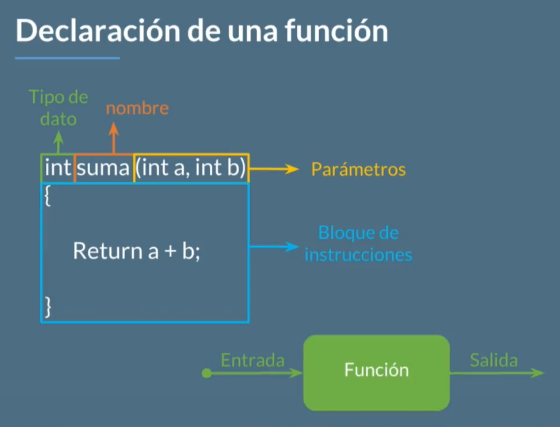
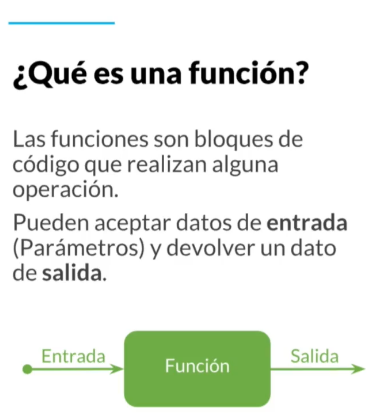
# Cadena de caracteres (string)

Generalmente se dice que es un arreglo de caracteres cuando lo que se almacenó son caracteres y no existe el carácter nulo al final.

Cuando el arreglo de caracteres termina con el carácter nulo se llama cadena de caracteres.

**Funciones: Divide y vencerás**

Las **funciones** son bloques de código que realizan alguna operación. Pueden aceptar datos de **entrada** (parámetros) y devolver un dato de **salida**.



Se pueden utilizar para:

* Encapsulamiento
* Reusabilidad de código
* Separar tareas
* Cambios a futuro

**Recursividad**

* En C, las funciones pueden llamarse a sí mismas.
* Si una expresión en el cuerpo de una función llama a la propia función, se dice que ésta es recursiva.
* La **recursividad** es el proceso de definir algo en términos de sí mismo y a veces se llama definición circular.

# 

**Apuntadores**

Un **apuntador** es una variable que guarda la dirección de memoria de otra variable.

Las direcciones de memoria se suelen describir como números en hexadecimal. Un **apuntador** es una variable cuyo valor es la dirección de memoria de otra variable. Se dice que un apuntador “apunta” a la variable cuyo valor se almacena a partir de la dirección de memoria que contiene el apuntador.

* El operador de dirección (&) regresa la dirección de una variable.
* El operador de indirección (\*) toma la dirección de una variable y regresa el dato que contiene esa dirección.

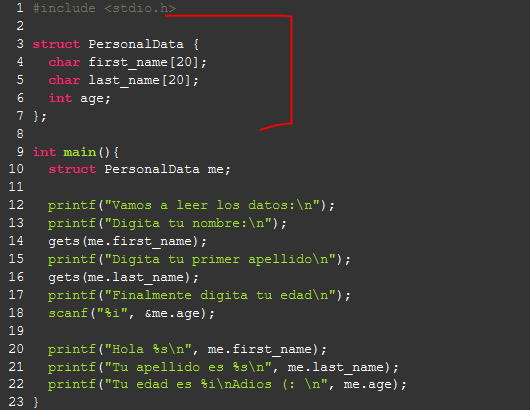
# 

# 

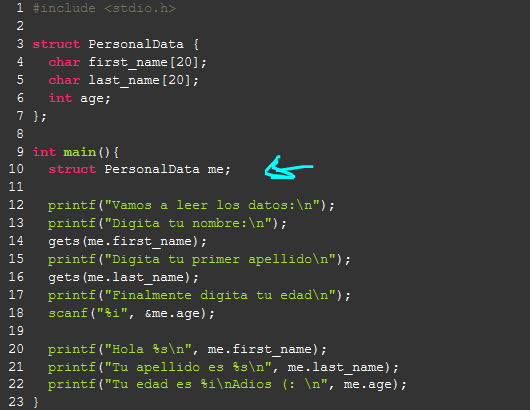
**Struct y manejo de archivos.**

Sabemos que en los arrays podemos almacenar un conjunto de información pero debe ser del mismo tipo. Sin embargo también tenemos la opción de guardar un conjunto de información pero esta vez de distinto tipo en algo llamado **Struct.**

**Para ello primero creamos nuestro struct asignando el tipo de datos a almacenar:**



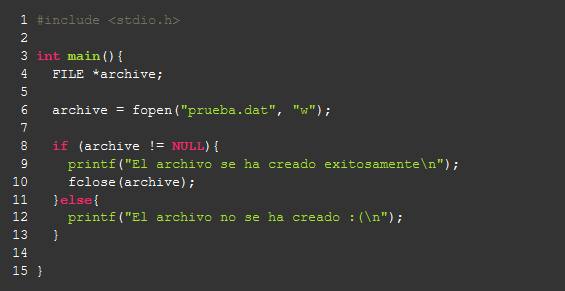
**Luego crearemos una variable** “me” **con nuestra estructura: *struct PersonalData* en ella podremos guardar la información como sus propiedades.**



# Escritura y lectura de archivos en C.

**Creación y apertura de archivos**  
Parámetros para la función fopen():

* ““rb””: Abre un archivo en modo binario para lectura, el fichero debe existir.
* ““w””: Abrir un archivo en modo binario para escritura, se crea si no existe o se sobrescribe si existe.



# 

# Manejo de librerías.

Una librería es código de programación ya escrito; un conjunto de funciones independientes para solucionar problemas concretos.

# 