

Tipos de Datos

bool	1 bit
int	4 byte
short	2 byte
float	4 byte
double	8 byte
long double	16 byte
char	1 byte

Operadores matemáticos

=	Asignación	&	AND (registro)
+	Suma		OR (registro)
-	Resta	^	XOR (registro)
*	Multiplicación	~	NOT (registro)
/	División	<<	Desplaz. bit izq
%	Módulo	>>	Desplaz. bit der

Funciones

void nombre () {}

Estas funciones **NO** devuelven valores, podemos verlas como un simple bloque de código que podemos repetir haciendo diversas llamadas

tipo_de_dato nombre () {}

Estas funciones **SI** devuelven valores, pueden ser de cualquier tipo (int, char, float ...). Su principal diferencia es que cuando hacemos las llamadas a dichas funciones en esa llamada tendremos un valor. ¿Qué valor? el que pongamos el **return**. ES OBLIGATORIO el return

Estructura de función

Funciones (cont)

Al declarar una función podemos hacerlo bien sin parámetros **paréntesis vacíos**, o bien podemos pasarle parámetros, los cuales necesitamos rellenar en la llamada **paréntesis (tipo_dato nombre)** podemos poner tantos parámetros como quedamos

! En el caso de las funciones con parámetros, podemos poner tantos parámetros como **tipo_dato nombre** pongamos

Comparadores

==	Igual a	!=	Distinto
<	Menor	>	Mayor
<=	Menor igual	>=	Mayor igual
&&	AND (lógico) Y		OR (lógico) O
!	NOT		

I/O de datos

Escribir **printf**("texto");
texto

Escribir int **printf**("texto %i", variable_int);

Escribir float **printf**("texto %f", variable_float);

Escribir double **printf**("texto %d", variable_double);

Escribir caracter **printf**("texto %c", variable_char);

Leer entero **scanf**("%i", &variable_almacen_int);

Leer float **scanf**("%f", &variable_almacen_float);

Leer double **scanf**("%d", &variable_almacen_double);

Leer caracter **scanf**("%c", &variable_almacen_char);

Aleatorios

Num aleatorio **rand()**%numero-
_limite

Num aleatorio **(rand())%(M-m+1)+m)**
rango

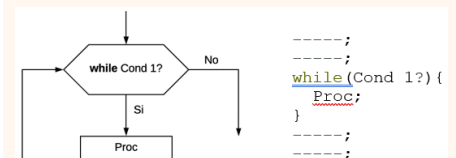
Modificar **seed** **srand**(time(NULL))

! En el primer caso el primer número es el 0

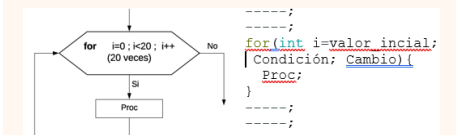
! En el segundo caso el M es el num mayor y la m el menor del rango

! srand crea un nuevo seed para que el num sea realmente aleatorio, debemos llamarlo al menos UNA vez en el **MAIN**

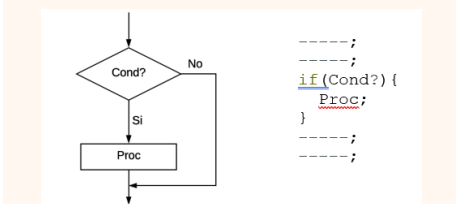
while



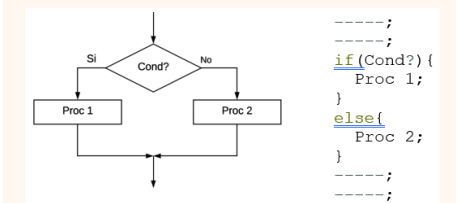
for



if



if -else



return

```
-----;
int valor=función();//llamada
-----;
int función() { //Declaración
    if (Cond?)
        return 100;
    else
        return 200;
}
```

Vectores

Se declaran con **tipo_dato nombre_vector[]**. Los vectores son arrays de dimensión 1, y el número que va a aparecer entre corchetes será la cantidad de elementos que va a tener **siempre** del mismo tipo de dato de que hemos creado el vector. Para obtener el valor de una posición **nombre_vector[posicion]**

! También se puede inicializar desde la declaración **tipo_dato nombre_vector[] = {elementos, por, comas}**

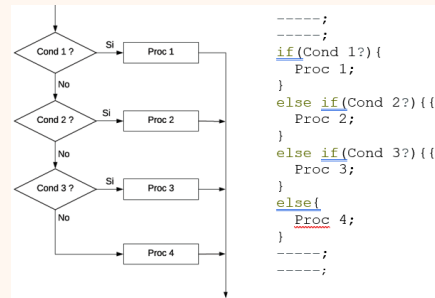
Arrays

Se declaran con **tipo_dato nombre_arrays[][]**. Los arrays es un conjunto de vectores (*normalmente de dimensión 2 ej: 5x5*), y el número que va a aparecer entre corchetes será la cantidad de elementos que va a tener **siempre** del mismo tipo de dato de que hemos creado el vector. En este caso en cada corchete irá la cantidad de elementos de cada columnas, y el número de filas.

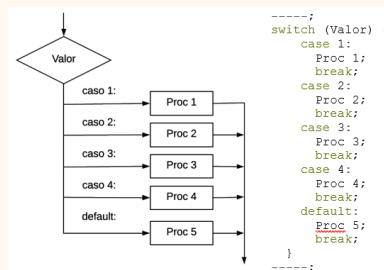
Para obtener el valor de una posición **nombre_vector[posicion_columna][posicion_fila]**

! También se puede inicializar desde la declaración **tipo_dato nombre_vector[] = {{elementos, por, comas}, {elementos, por, comas}}**

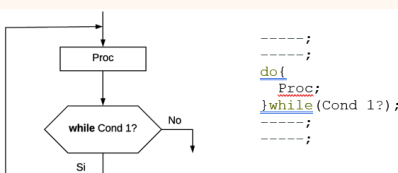
if-else if- else



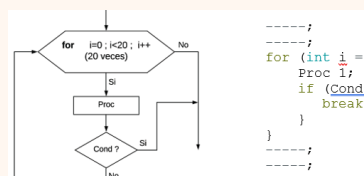
switch



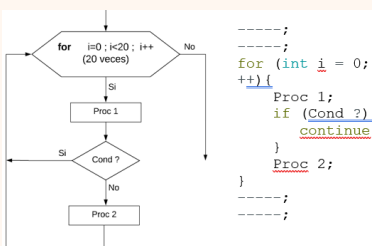
do-while



Elementos de control extra - break



Elementos de control extra - continue



Recorrer arrays

```
for (int i = punto_inicio;
i <= limite_elementos; i++) {
    for (int j = punto_inicio;
j <= limite_elementos; j++) {
        printf ("%i \t", vector [i] [j]);
    }
}
```

! Con esto imprimimos cada posición del array, indicando donde empezamos y acabamos, y avanzando de 1 en 1. **debemos hacer un for para cada dimensión, i para las columnas y j para las filas.** Para recorrerla de principio a fin de 1 en 1 un vector de dim 9x9 **for (int i=0; i<=8; i++)** y dentro de ese for **for (int j=0; j<=8; j++)**

Recorrer vectores

```
for (int i = punto_inicio;
i <= limite_elementos; i++) {
    printf ("%i \t",
vector [i]);
}
```

! Con esto imprimimos cada posición del vector, indicando donde empezamos y acabamos, y avanzando de 1 en 1. Para recorrerla de principio a fin de 1 en 1 un vector de dim 9 **for (int i=0; i<=8; i++)**

Volver a pedir datos sin cerrar el programa

Una de las opciones más claras a la hora de reiniciar la petición de entrada de datos al usuario sin cerrar el programa es usar un do-while.

En el **do** metemos el programa general con la evaluación de la entrada así como con la 1a petición de datos; en el **while**, ponemos como condición que no se cumpla ninguna de las válidas, y dentro de él otra vez a pedir datos.