02

ESTRUCTURAS DE CONTROL

Condicionales

Simples

if

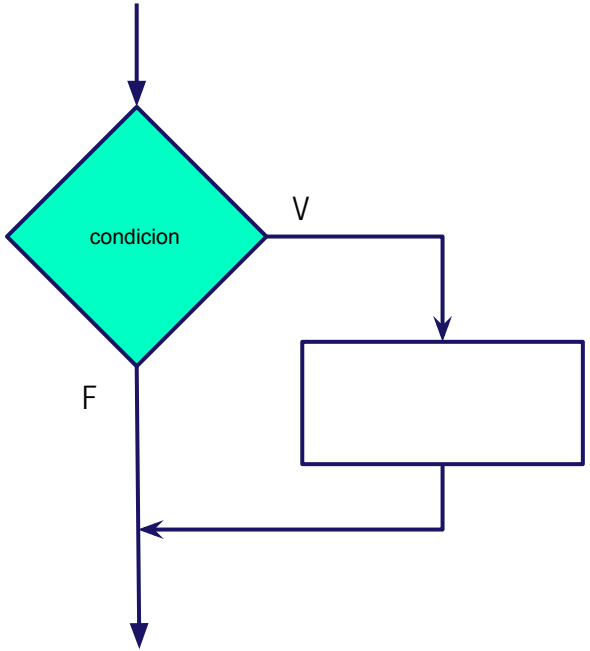
Dobles

if ... else

Múltiples

switch

Condicional simple




```
if (condicion)
    proceso();
```

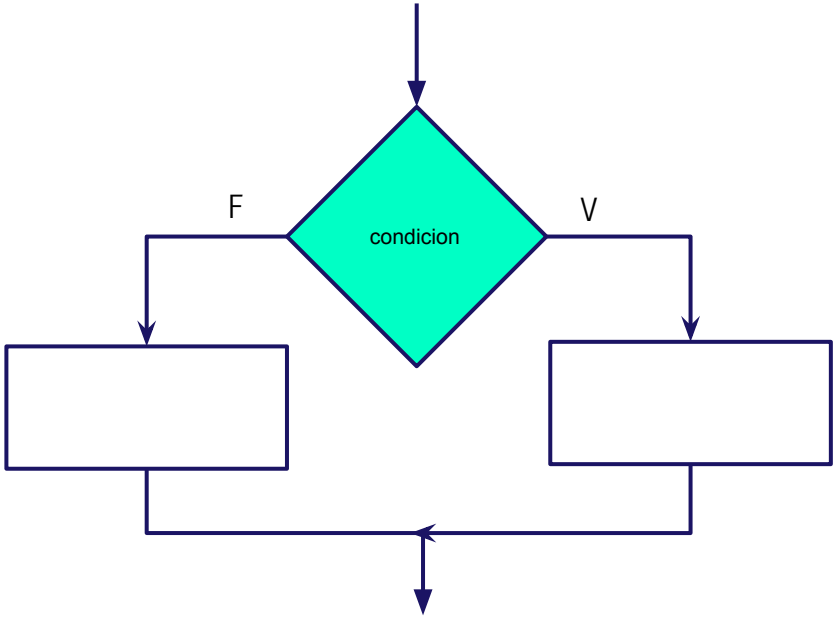
Cuando sólo hay un proceso dentro del if podemos omitir las llaves

```
if(condicion)
{
    proceso1();
    proceso2();
    proceso3();
}
```

Para escribir condiciones utilizamos operadores relacionales, por ejemplo: num<10

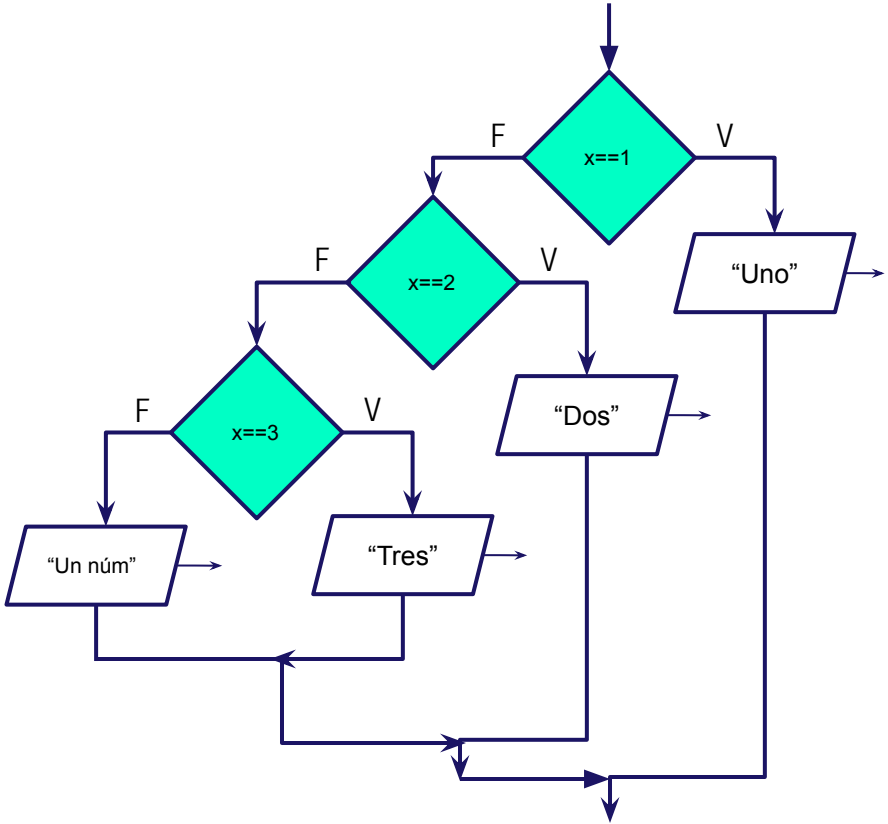
Para combinar condiciones utilizamos operadores lógicos, por ejemplo: num>0 && num<10

Condicional doble




```
if (condicion)
        proceso();
else
        proceso2();

if(condicion)
{
        proceso1();
        proceso2();
}
else
{
        proceso3();
}
```

Condicional múltiple



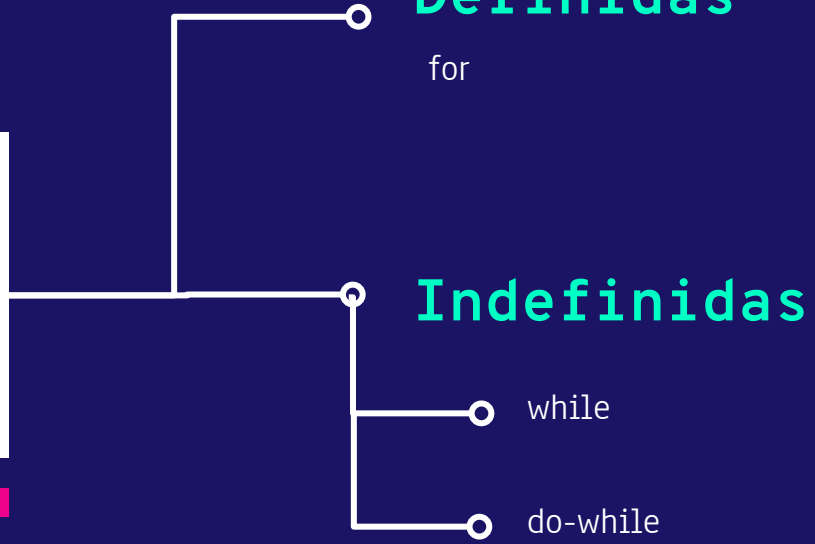

```
int x;
scanf("%i", &x);
switch(x)
{
        case 1:
                printf("Uno");
                break;
        case 2:
                printf("Dos");
                break;
        case 3:
                printf("Tres");
                break;
        default:
                printf("Un número");
}
```

Biblioteca ctype.h

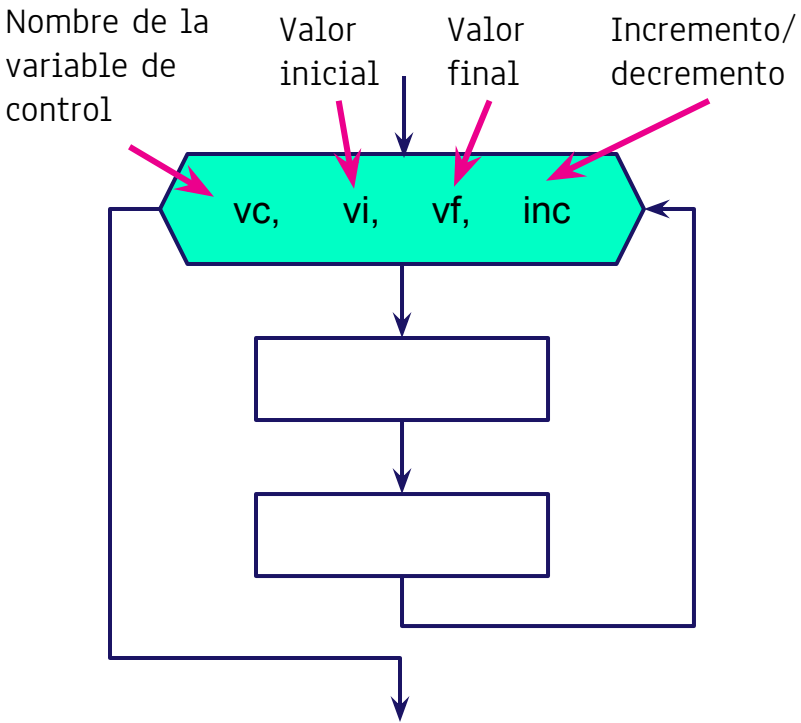
Permite realizar comprobaciones y manipulaciones de caracteres.

Función	Descripción
int **isalnum**(int c)	Determina si el carácter es alfanumérico (A-Z, a-z, 0-9).
int **isdigit**(int c)	Determina si el carácter es un dígito (0-9).
int **islower**(int c)	Determina si el carácter es una minúscula (a-z).
int **isupper**(int c)	Determina si el carácter es una mayúscula (A-Z)
int **isspace**(int c)	Determina si un caracter es un espacio en blanco.
int **toupper**(int c)	Convierte una letra minúscula a mayúscula
int **tolower**(int c)	Convierte una letra mayúscula a minúscula

Estructuras iterativas



Ciclo For



Variable de control

Valor inicial

Incremento

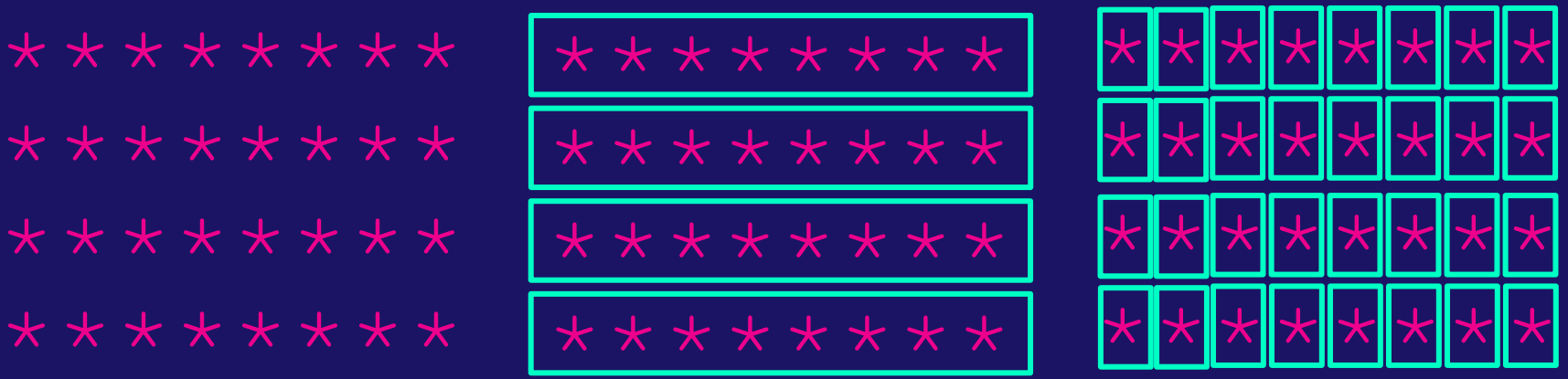
```
int i;

for(i=0; i<=10; i++)
{
        printf("%i", i);
        printf("-\n");
}

for(i=10; i>=0; i--)
{
        printf("%i", i);
        printf("-\n");
}
```

Ejemplo: Dibujando un rectángulo

¿Cómo le hago?

Ejemplo: Dibujando un rectángulo

Primero hay que descomponer el problema en problemas más pequeños.



Ejemplo: Dibujando un rectángulo

Primero hay que descomponer el problema en problemas más pequeños.

Problema 1: ¿Cómo dibujar un asterisco en la pantalla?

Problema 2: ¿Cómo dibujar una fila de asteriscos en la pantalla?

Problema 3: ¿Cómo repito esta fila varias veces?

Ejemplo: Dibujando un rectángulo

Para resolver cada problema debemos **encontrar patrones** y utilizar las herramientas del lenguaje de programación

¿Cómo dibujar un asterisco en la pantalla?

```
printf("*");
```

¿Cómo dibujar una fila de asteriscos en la pantalla?

Si la fila es de un número constante de asteriscos.

```
printf("********");
```

Si la fila es de un número variable de asteriscos, se pueden utilizar estructuras de control.

```
for(i=1; i<=8; i++)
    printf("*");
```

¿Cómo repito esta fila varias veces?

Ejemplo: Dibujando un rectángulo

La palabra patrón hace referencia a una repetición.

- Los asteriscos en cada fila se **repiten** un número determinado de veces.

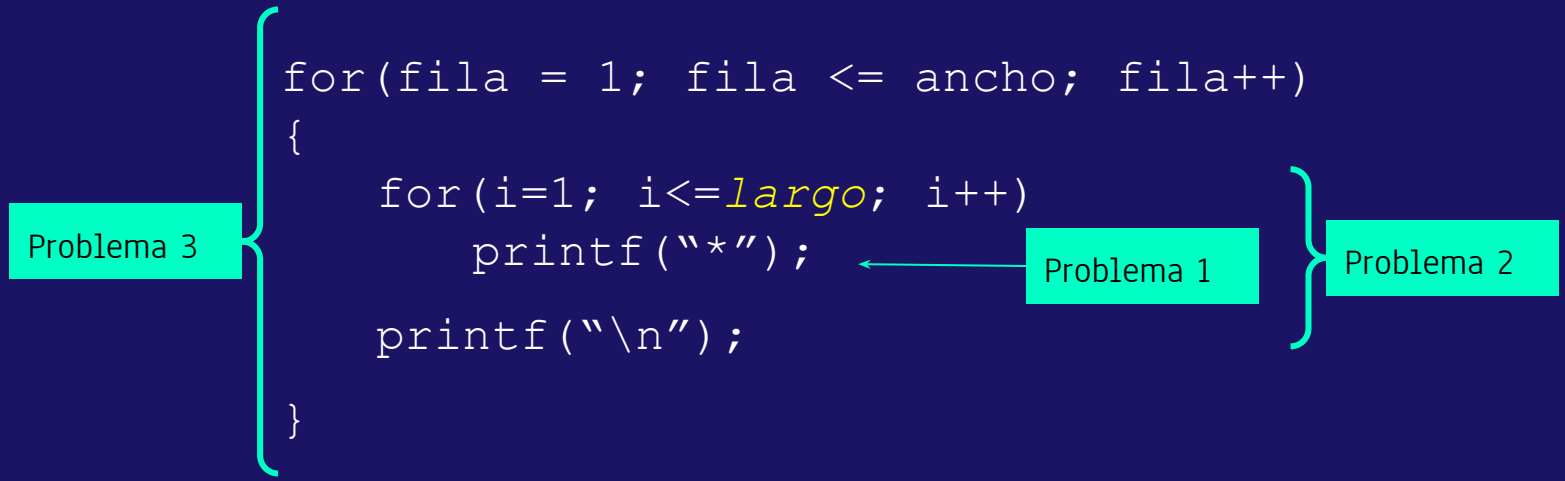
```
for(i=1; i<=largo; i++)
    printf("*");
```

- Al final de **cada** fila se imprime un salto de línea.

```
for(i=1; i<=largo; i++)
    printf("*");

printf("\n");
```

Ejemplo: Dibujando un rectángulo

- Cada fila se repite un número determinado de veces.



Ejemplo: Dibujando un rectángulo

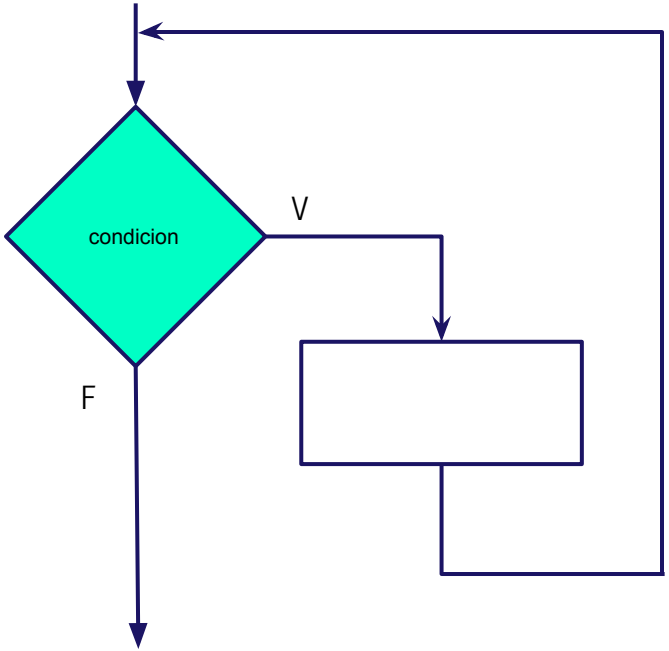
¿Cómo le hago?

Ejemplo: Dibujando un rectángulo

Encontremos patrones:

1. La primera y última fila están rellenas, las demás son huecas.
2. En las filas rellenas se coloca un número de asteriscos igual al largo del rectángulo.
3. En las filas huecas se coloca un número de espacios igual al largo del rectángulo menos dos.
4. En todas las filas se termina con un salto de línea.

Ciclo While




```
int n = 0;
while (n <= 10)
{
        printf("%i\n", n);
        n++;
}
n = 10;
while (n >= 0)
{
        printf("%i\n", n);
        n--;
}
```

condición

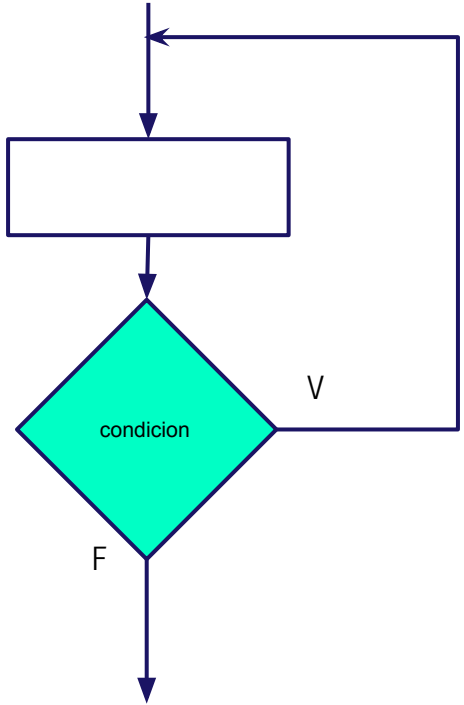
Ciclo while

Un ciclo while se puede controlar básicamente de dos maneras.

Por contador	Por centinela o bandera
Se conoce el número de veces que se desea ejecutar el ciclo. Se requiere de una variable (contador) cuyo valor se incremente dentro del ciclo.	Se desconoce cuántas veces se ejecutará el ciclo. Una variable (centinela) determinará cuándo se dejará de ejecutar el ciclo (si esta variable tiene solo dos posibles valores se llama bandera).

`int n = 0;`
 	
`while (n <= 10)`
`{`
`  printf("%i\n", n);`
`  n++;`
`}`
	`char res = 'S';`
	
	`while(res == 'S' || res == 's')`
	`{`
	`  printf("¿Quieres continuar");`
	`  scanf("%c", &res);`
	`}`

Ciclo Do-While




```
int n;
do
{
      printf("Dame un número");
      scanf("%i", &n);
}
while(n%2==0);
```

Punto y coma

condición


while (not edge) {
run();
}
do {
run();
} while (not edge);

Programación estructurada

La programación estructurada se basa en la técnica de diseño descendente (*top-down*).

En el diseño descendente el problema se descompone en pequeños problemas. Los subproblemas se descompondrán hasta tener un algoritmo detallado.

Corrado Bohm y Giuseppe Jacopini publicaron en 1966 el artículo *Flow diagrams, Turing machines and languages with only two formation rules*. En él demostraban la posibilidad de expresar cualquier algoritmo como la combinación de estructuras lógicas básicas.


Computational Linguistics

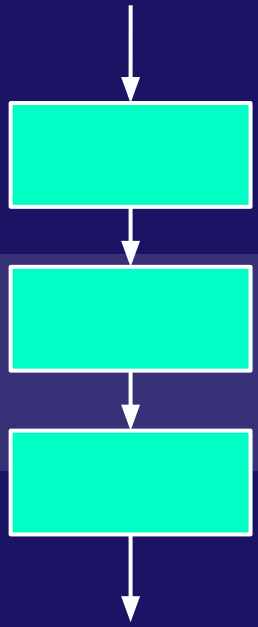
D. G. BOBROW, Editor

Flow Diagrams, Turing Machines And Languages With Only Two Formation Rules

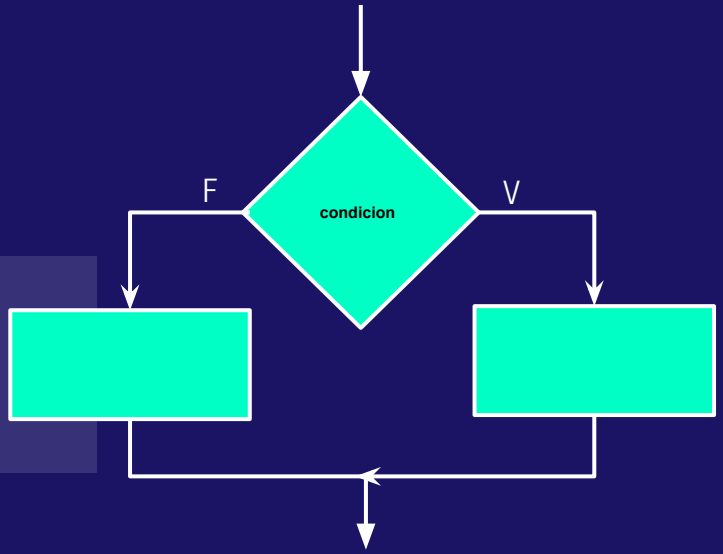
Corrado Bohm and Giuseppe Jacopini
Instituto di Calcoli Numerici and Istituto Nazionale per le Applicazioni del Calcolo, Rome, Italy

Estructuras elementales

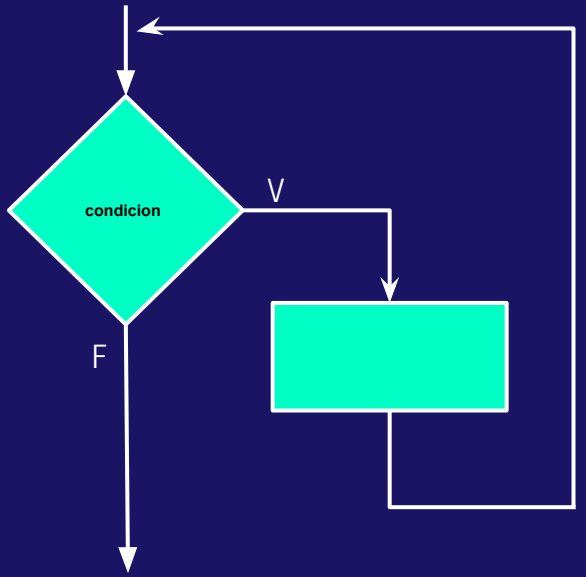
Cada subproblema se resuelve con la construcción de códigos a partir de **tres estructuras lógicas básicas**:



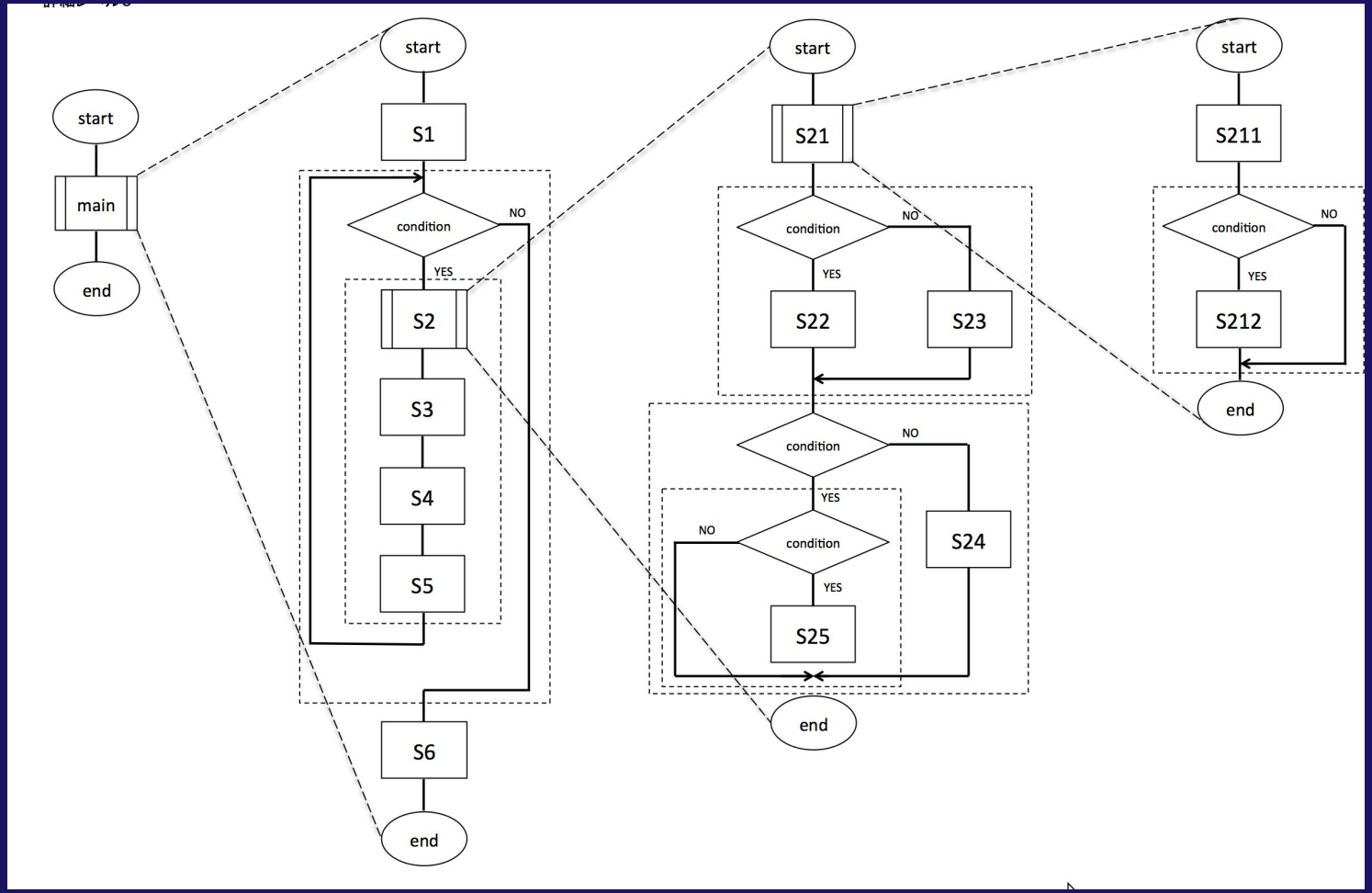
Secuencia



Selección
(condicionales)



Repetición


start
main
end
start
S1
condition
NO
YES
S2
S3
S4
S5
S6
end
start
S21
condition
NO
YES
S22
S23
condition
NO
YES
S24
NO
condition
YES
S25
end
start
S211
condition
NO
YES
S212
end

Un programa estructurado se puede segmentar en pequeños subprogramas para simplificar la lectura del código y de los diagramas de flujo.

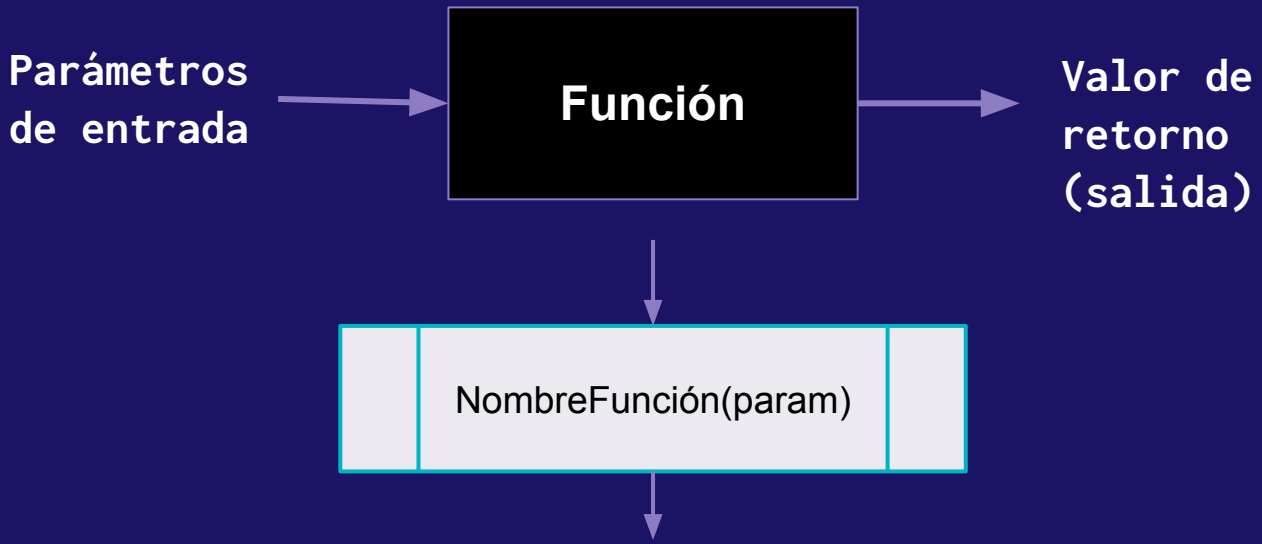
En lenguaje C podemos lograr esto a través de funciones.

Funciones

Definidas por el programador

¿Qué es una función?

Una función es un fragmento de código independiente que ejecuta una tarea específica. Una función puede recibir valores (parámetros de entrada) y también puede regresar un valor (valor de retorno).



Definición de una función

Firma de la función

Valor de retorno (salida)
Sólo se indica el tipo de dato que devuelve

Parámetros de entrada
Se indica el tipo y el nombre de cada parámetro

```
tipo_de_retorno nombre_funcion (tipo1 nombre_variable1, tipo2 nombre_variable2, ...)
{
    instruccion1;
    instrucction2;

    return dato_a_retornar;
}
```

Cada definición de parámetro se separa con una coma

La lista de parámetros de entrada se encierra entre paréntesis.

Palabra que indica que se regresará un valor.

Valor o nombre de la variables cuyo valor va a ser regresado por la función

Ejemplo: Función saluda

```
#include <stdio.h>

void saluda()
{
    printf("Hola");
}

int main (void)
{
    saluda();

    return 0;
}
```

Definición de la función *saluda*

Llamada de la función *saluda*

Definición de la función principal

Ejemplo: Función mensaje

```
#include <stdio.h>

void mensaje(int tipo_msj)
{
    if(tipo_msj == 1)
        printf("Hola");
    else
        printf("Adiós");
}

int main (void)
{
    mensaje(1);
    mensaje(0);
    return 0;
}
```

Definición de la función *mensaje*

Llamada a la función *mensaje* con diferentes valores para el parámetro tipo_msj.

Ejemplo: Función suma

```
#include <stdio.h>

int suma(int n1, int n2)
{
    int suma = n1+n2;
    return suma;
}

int main (void)
{
    int res;
    res = suma(4,5);
    printf("El resultado es %i", res);
    return 0;
}
```

Definición de la función *suma*

Llamada a la función suma con valores 4 y 5 para los parámetros n1 y n2.

¿Por qué utilizamos una función?

1. **Diseño descendente**

   El problema se puede dividir en subproblemas y cada subproblema puede ser resuelto en una función.

2. **Reutilización de código**

   Las funciones se pueden llamar donde se necesiten sin necesidad de volver a escribir el código.

3. **Ocultamiento de información**

   Las funciones se encargan de realizar la tarea, el programador sólo llama a la función para que la realice.

4. **Facilidad de detección y solución de errores**