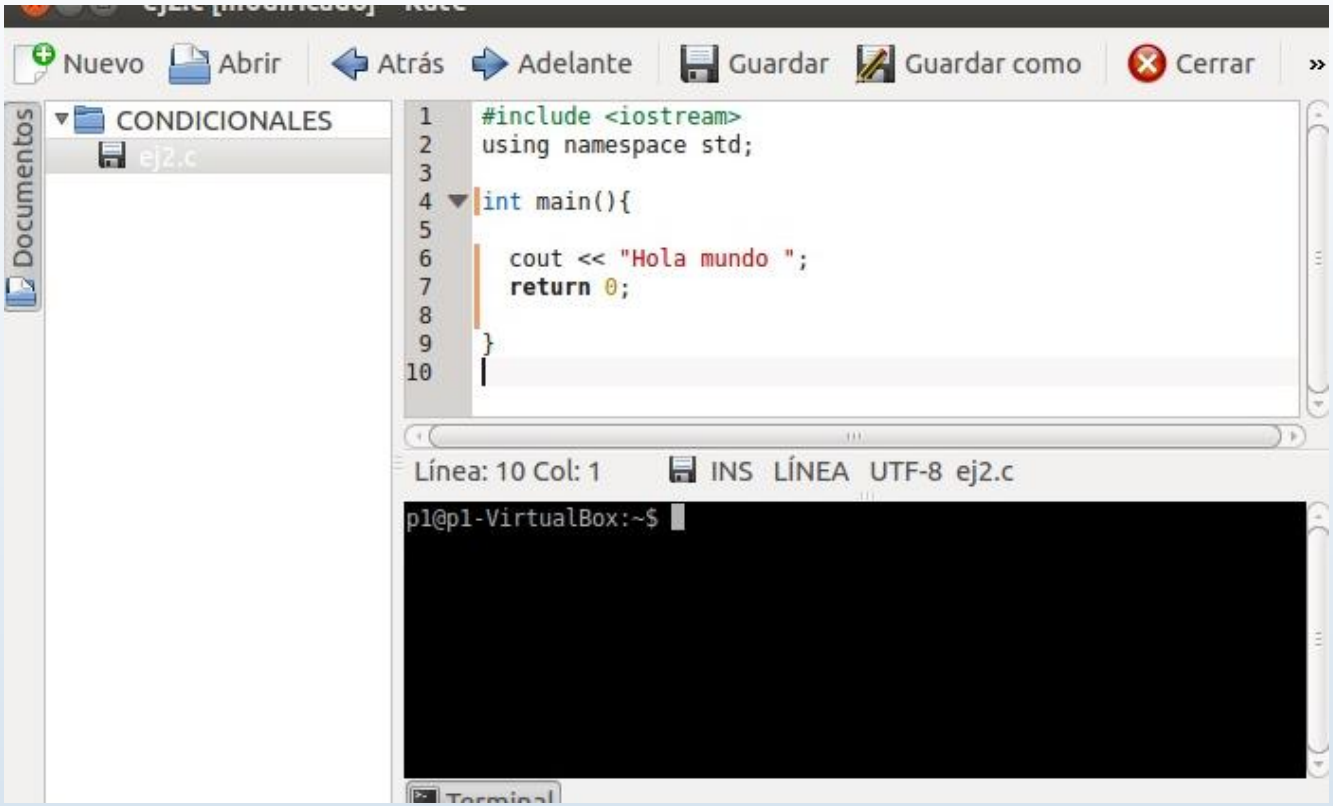


Programación 1

Programación modular

Grado en Ingeniería Informática

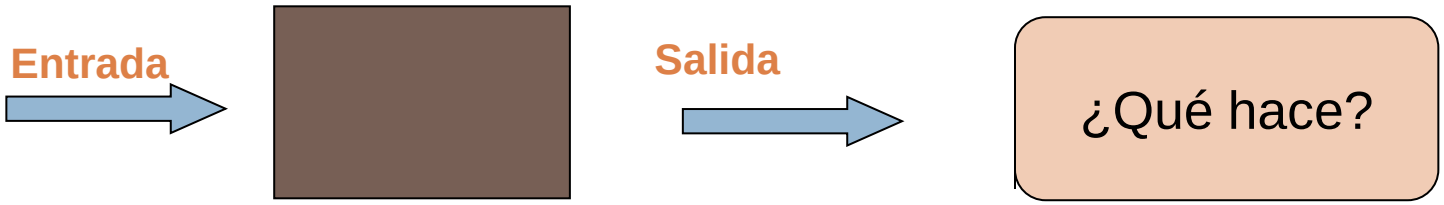
Concepto de Módulo

- Cuando un programa es grande y complejo no es conveniente que todo el código esté dentro del programa principal (función main() en lenguaje C) Un módulo o subprograma …
  - es un bloque de código que se escribe aparte del programa principal
  - se encarga de realizar una tarea concreta que resuelve un problema parcial del problema principal
  - puede ser invocado (llamado) desde el programa principal o desde otros módulos
  - permite ocultar los detalles de la solución de un problema parcial (caja negra)

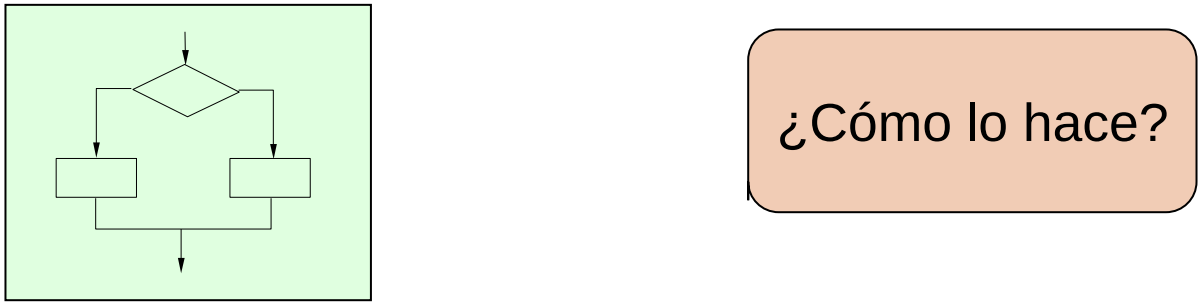
Caja negra

- Cada módulo es una caja negra para el programa principal o para el resto de módulos

- Para utilizar un módulo desde el programa principal o desde otros módulos …
  - Necesitamos conocer su interfaz, es decir, sus entradas y salidas



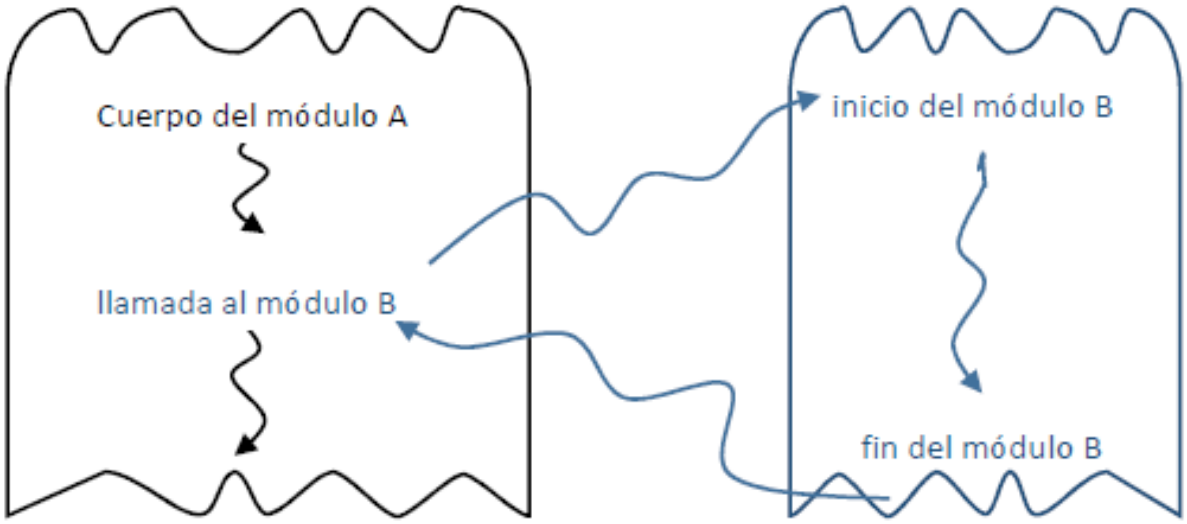
  - No necesitamos conocer los detalles internos de funcionamiento



Transferencia del flujo de control

- Cuando un módulo A llama (invoca) a otro módulo B, el flujo de control (flujo de ejecución) pasa al módulo B

- Cuando termina de ejecutarse el módulo B, el flujo de control continúa en el módulo A, a partir de la sentencia siguiente a la llamada del módulo B



Función

- Es un módulo que devuelve un único resultado asociado a su nombre.
- Recibe unos datos de entrada y en base a ellos, genera y devuelve un resultado.

Ejemplos: las funciones matemáticas

Raíz cuadrada (9) → 3

Potencia (4, 3) → 64

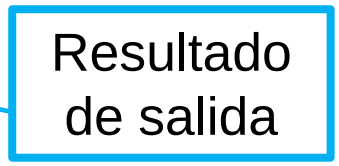
Log (4) → 0.60206

Cos (∏) → -1

Función: ejemplo

Función que recibe dos números y devuelve el más grande de los dos.

```
int max (int a, int b){
     int m;  //variable local
     if (a>b)
          m=a;
     else
          m=b;
     return (m);
}
```



Usando funciones

- Para ejecutar una función hay que realizar una llamada desde el main o desde otro módulo.
- Para llamar a una función se usa su nombre poniendo entre paréntesis los parámetros de entrada separados por comas.
- Si la función no tiene parámetros se ponen los paréntesis vacíos.
- Cuando se llama a una función, se obtiene un valor como resultado. **Es muy importante que la llamada se incluya en una expresión que permita aprovechar el resultado.**

Usando funciones: ejemplo


```
#include <iostream>
using namespace std;

int main() {
      int n1, n2;
      int mayor; //el mayor número de los 2 introducidos
      cout << "Introduce dos números enteros:";
      cin >> n1 >> n2;
      mayor = max(n1, n2);
      cout << "El mayor número es:" << mayor;
      return 0;
}
```

Llamando a la función y guardando el resultado en la variable mayor

Sobre la sentencia return

- Finaliza la ejecución del cuerpo de la función.
- Se encarga de devolver el valor de retorno de la función, después de evaluar su *expresión* asociada.
- Es recomendable usar una sola sentencia return dentro del cuerpo de una función.
- Debería ser la última sentencia del cuerpo de la función.

return (*expresión***);**

Procedimiento

- Es un módulo que realiza una tarea específica
- Puede recibir cero o más valores y devolver cero o más valores a través de la lista de parámetros.

```
void autor() {
      cout << "**********************\n”;
      cout << "* Realizado por P1 *\n";
      cout << "**********************\n";
}
```

Este procedimiento muestra un mensaje en pantalla y no tiene parámetros

Usando procedimientos

- Para ejecutar un procedimiento hay que realizar una llamada desde el main o desde otro módulo.
- Para llamar a un procedimiento se usa su nombre poniendo entre paréntesis los parámetros separados por comas.
- Si el procedimiento no tiene parámetros se ponen los paréntesis vacíos.

```
int main() {
        …
        autor();
        …
        …
        return 0;
}
```

Estructura de un programa en C

#directivas del preprocesador

Declaración de constantes

Declaración de procedimientos y funciones — Prototipos

main() {

Declaración de variables (de tipos simples)

Cuerpo principal

sentencias de control

llamadas a procedimientos y funciones

}

Definición de procedimientos y funciones

Transferencia de información

- La transferencia de información entre módulos se realiza a través del paso de parámetros o argumentos.
- Un parámetro permite pasar información desde un módulo a otro y viceversa.
- Un parámetro puede ser considerado como una variable cuyo valor debe ser o bien proporcionado por el programa principal al módulo o ser devuelto desde el procedimiento hasta el programa principal.
- Hay tres tipos de parámetros:
  - Entrada: sus valore son proporcionados por el módulo que llama
  - Salida: sus valores se calculan en el subprograma y se devuelven al módulo que ha llamado
  - Entrada/salida: proporcionan valores al módulo y éstos además pueden ser modificados.

Parámetros actuales y formales

- Parámetros actuales o reales
  - Los que aparecen en la sentencia de llamada al módulo

Nombre_del_módulo (pr1, pr2, …, prN)

```
mayor =  maximo(n1, n2);
```

- Parámetros formales o ficticios
  - Los que aparecen en la declaración del módulo

Nombre_del_módulo (tipo1 pf1, tipo2 pf2, …, tipoN pfN)

```
int maximo(int a, int b);
```

- **Correspondencia entre parámetros** actuales y formales:

✓ número de parámetros
✓ tipo de parámetros
✓ orden de los parámetros
✘ ~~nombre de los parámetros~~

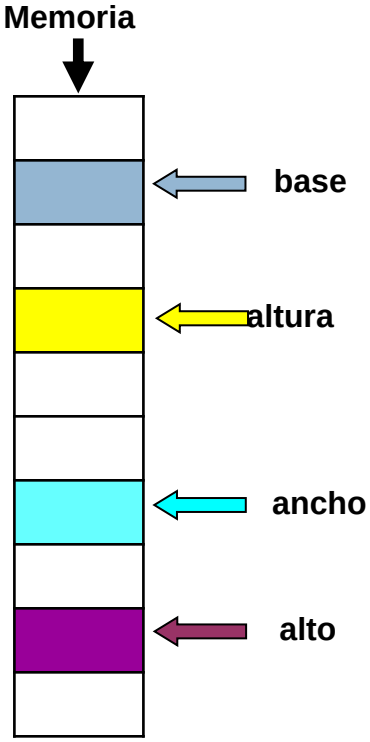
Paso de parámetros por Valor

- El módulo recibe una copia del valor del dato (parámetro actual) que el módulo invocador le pasa
- El parámetro actual puede ser cualquier expresión evaluable en el momento de la llamada al módulo
- Si dentro del módulo se modifica el parámetro formal correspondiente, el valor del parámetro actual permanece inalterable

```
main() {
  int  base, altura, area, perimetro;

  cout << "Díme la base del rectángulo:";
  cin >> base;
  cout << "Díme su altura:";
  cin >> altura;

  rectangulo(base, altura, area, perimetro);

  cout << "Area: " << area << endl;
  cout << "Perímetro: " << perimetro;
  cout  << endl;
}
```

Paso de parámetros por Valor

```
void rectangulo( int ancho, int alto,
                 int &area_rect, int &perim )
{
  area_rect = ancho * alto;
  perim = 2 * (ancho + alto);
}
```



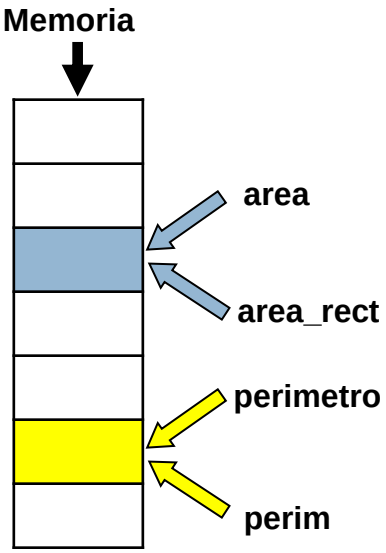
Paso de parámetros por Referencia

- El módulo recibe la referencia a la posición de memoria donde se encuentra dicho valor (dirección de memoria de una variable)
- El parámetro actual debe ser obligatoriamente una variable (que puede contener o no un valor)
- Si dentro del módulo se modifica el parámetro formal correspondiente, se estará cambiando el contenido en memoria del parámetro actual

```
main() {
   int  base, altura, area, perimetro;

   cout << "Díme la base del rectángulo:";
   cin >> base;
   cout << "Díme su altura:";
   cin >> altura;

   rectangulo(base, altura, area, perimetro);

   cout << "Area: " << area << endl;
   cout << "Perímetro: " << perimetro;
   cout  << endl;
}
```

Paso de parámetros por Referencia

```
void rectangulo( int ancho, int alto,
                 int &area_rect, int &perim )
{
   area_rect = ancho * alto;
   perim = 2 * (ancho + alto);
}
```



Concepto de ámbito de una variable

El ámbito de una variable define la visibilidad de la misma, es decir, desde dónde se puede acceder a dicha variable

```
main() {
  int   n;  // número introducido por teclado  (dato de entrada)
   cout << "Introduce un número entero:  ";
   cin >> n;
   if (es_primo(n))
     cout << "El número es primo";
   else
     cout << "El número no es primo";
   cout << endl;
}
```

ámbito de ***n***

```
// Este módulo comprueba si un número es primo o no
bool es_primo(int  num)
{
  int  cont;  // contador  (dato auxiliar)
  bool  primo;  // es primo o no (dato de salida)
   primo = true;
   cont = 2;
   while ( (cont < num) && primo) {
     // comprobar si es divisible por otro número
     primo = ! (num % cont == 0);
     cont = cont + 1;
   }
   return (primo);
}
```

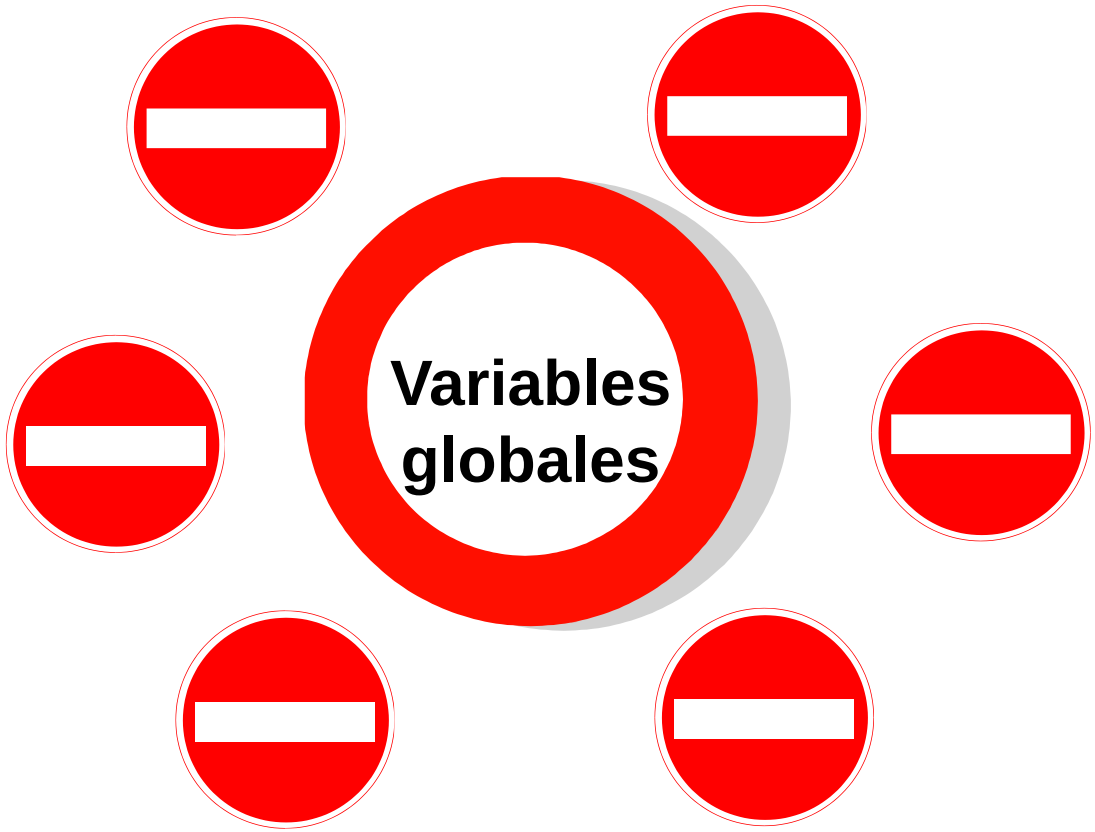
ámbito de
num, cont, primo

Variables locales y variables globales

- Variable local
  - Su ámbito es el cuerpo del módulo en donde está declarada
  - Se crea cuando se declara y se destruye cuando finaliza la ejecución del módulo
- Variable global
  - Su ámbito es todo el programa (todos sus módulos y el programa principal)
  - Se crea cuando se declara y se destruye cuando finaliza la ejecución del programa

Prohibido utilizar variables globales

La **comunicación** entre módulos debe realizarse a través de parámetros, y **NO de variables globales**



Ventajas de la programación modular

- Facilita el diseño descendente y la programación estructurada
- Reduce el tiempo de programación
  - *Reusabilidad* : estructuración en *librerías* específicas (biblioteca de módulos)
  - División de la tarea de programación entre un equipo de programadores
- Disminuye el tamaño total del programa
  - Un módulo sólo esta escrito una vez y puede ser utilizado varias veces desde distintas partes del programa
- Facilita la detección y corrección de errores
  - Mediante la comprobación individual de los módulos
- Facilita el mantenimiento del programa
  - Los programas son más fáciles de modificar
  - Los programas son más fáciles de entender (más legibles)

Bibliotecas del lenguaje C / C++

- La mayoría de lenguajes de programación proporcionan una colección de procedimientos y funciones de uso común (**bibliotecas** o **librerías**)
- En lenguaje C / C++, para hacer uso de los módulos incluidos en una biblioteca se utiliza la directiva del compilador ***#include***
- Existe una gran variedad de bibliotecas disponibles:
  - Funciones matemáticas
  - Manejo de caracteres y de cadenas de caracteres
  - Manejo de entrada y salida de datos
  - Manejo del tiempo (fecha, hora, …)
  - etc.

Algunas funciones predefinidas en lenguaje C / C++

Librería C++	Librería C	Función	Descripción
<math.h>	<math.h>	double cos(double x)	Devuelve el coseno de x
		double sin(double x)	Devuelve el seno de x
		double exp(double x)	Devuelve e^x
		double fabs(double x)	Devuelve el valor absoluto de x
		double pow(double x, double y)	Devuelve x^y
		double round(double x)	Devuelve el valor de x redondeado
		double sqrt(double x)	Devuelve la raiz cuadrada de x
<iostream>	<ctype.h>	int isalnum(int c)	Devuelve verdadero si el parámetro es una letra o un dígito
		int isdigit(int c)	Devuelve verdadero si el parámetro es un dígito
		int toupper(int c)	Devuelve el carácter en mayúsculas
	<stdlib.h>	int rand(void)	Devuelve un número aleatorio entre 0 y RAND_MAX

Librería C++	Librería C	Constantes	Descripción
<iostream>	<stdint.h>	INT_MIN	Menor número entero representable
		INT_MAX	Mayor número entero representable

Ejercicios

1. Hacer una función que devuelva la letra que le corresponde a un número de DNI que se pasa como parámetro mediante el siguiente algoritmo:
    1. Calcular el resto de la división del DNI entre 23
    2. En función del valor del resto, asociar la letra correspondiente según la siguiente tabla:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	R	W	A	G	M	Y	F	P	D	X	B	N	J	Z	S	Q	V	H	L	C	K	E

2. Diseña un módulo que reciba como parámetro un número `n` y dibuje en pantalla un cuadrado de tamaño `n` formado por asteriscos.
3. Mejora el ejercicio 2 añadiendo otro parámetro que permita que el cuadrado se dibuje con el carácter enviado como parámetro.
4. Diseña un módulo que reciba dos variables e intercambie los valores de las mismas.
5. Diseña un módulo que permita leer y validar un dato de entrada de manera que su valor sea mayor que 0 y menor que 100 y devuelva la suma y la cuenta de los números entre 1 y dicho valor.