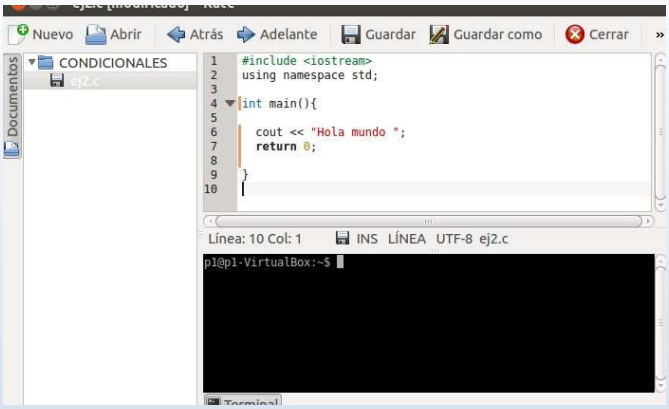


Programación 1

Tema 4. Programación modular

Grado en Ingeniería Informática

Objetivos / Competencias

1. Utilizar el diseño descendente para resolver problemas de relativa complejidad
2. Comprender las diferencias entre procedimientos y funciones
3. Saber modularizar programas en lenguaje C

Índice

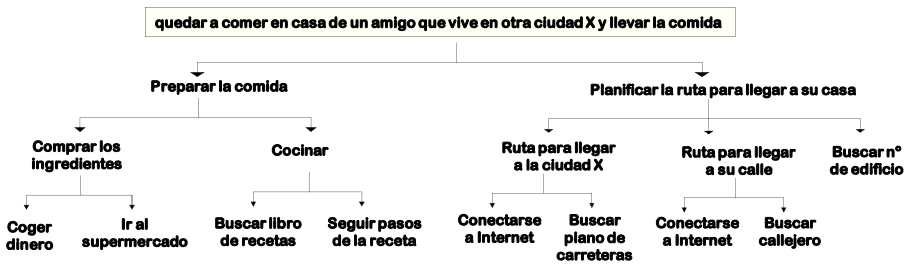
3

1. Descomposición modular
2. Comunicación entre módulos
3. Procedimientos y Funciones
4. Ámbito de las variables
5. Estructura general de un programa
6. Funciones predefinidas en lenguaje C
7. Fuentes de información

Diseño descendente ("*top-down*")

4

- Para resolver un problema se divide en problemas más pequeños (subproblemas)
  - La descomposición del problema se realiza en una serie de niveles o pasos sucesivos de refinamiento, que forman una estructura jerárquica
  - Cada nivel de la jerarquía incluye un mayor nivel de detalle



Concepto de Módulo

5

- 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)

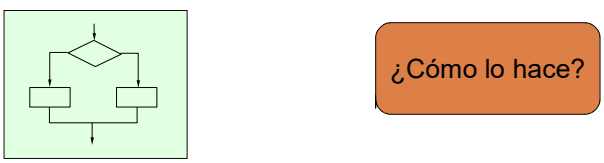
Caja negra

6

- 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



Ejemplo de módulos

7

```
main() {
  int   n1, n2;  // números introducidos por teclado  (datos de entrada)
  int   mayor;  // el mayor número de los 2 introducidos (dato de salida)
  int   menor;  // el menor número de los 2 introducidos (dato de salida)

  cout << "Introduce dos números enteros:  ";
  cin >> n1 >> n2;
  mayor = maximo(n1, n2);
  menor = minimo(n1, n2);
  cout << "El mayor número es:" << mayor;
  cout << "El menor número es:" << menor;
  cout << endl;
}
```

¿Qué hace el módulo "maximo()"?

¿Cómo lo hace?

```
// Este módulo devuelve el menor de dos números
int minimo(int a, int b)
{
  int  m;  // el menor de dos números (dato de salida)

  m = a;
  if (b < m)
    m = b;
  return(m);
}
```

```
// Este módulo devuelve el mayor de dos números
int maximo(int a, int b)
{
  int  m;  // el mayor de dos números (dato de salida)

  if (a > b)
    m = a;
  else
    m = b;
  return(m);
}
```

Declaración, definición y llamada de un módulo

8

Declaración del módulo

Nombre_del_módulo (declaración_de_parámetros)

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

Definición del módulo

Nombre_del_módulo (declaración_de_parámetros)

Declaración_de_variables_locales

Cuerpo del módulo: sentencias ejecutables

Fin_del_módulo

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

  if (a > b)
    m = a;
  else
    m = b;
  return(m);
}
```

Llamada del módulo

Nombre_del_módulo (lista_de_parámetros)

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

Ventajas de la programación modular

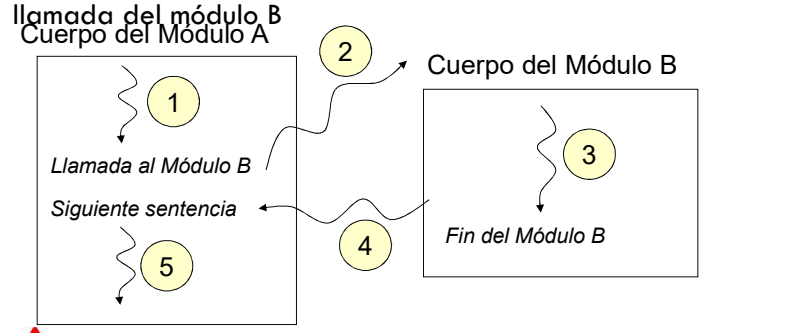
9

- 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)

Transferencia del flujo de control

10

- 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

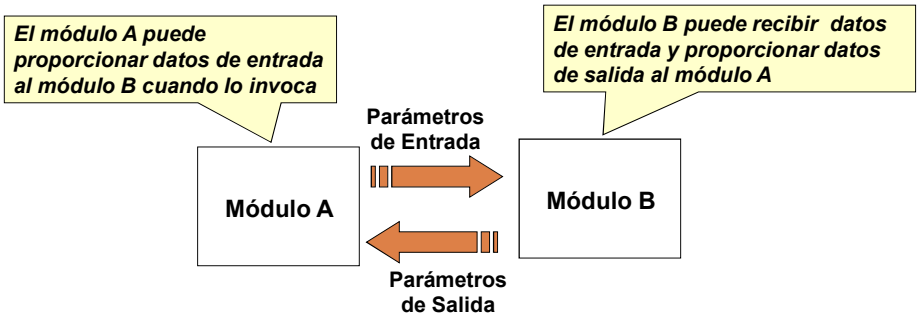


El programa principal puede ser considerado como un módulo que puede invocar a otros módulos, pero que no puede ser invocado por ningún módulo

Transferencia de información

11

- La transferencia de información entre módulos se realiza a través del paso de parámetros o argumentos
- Un módulo puede tener parámetros de entrada y/o de salida



Parámetros actuales y formales

12

- 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

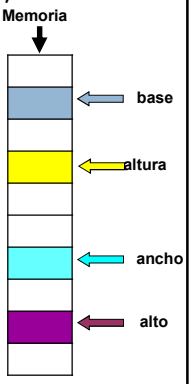
13

- 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

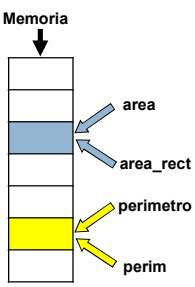
14

- 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 <u>obligatoriamente</u> 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

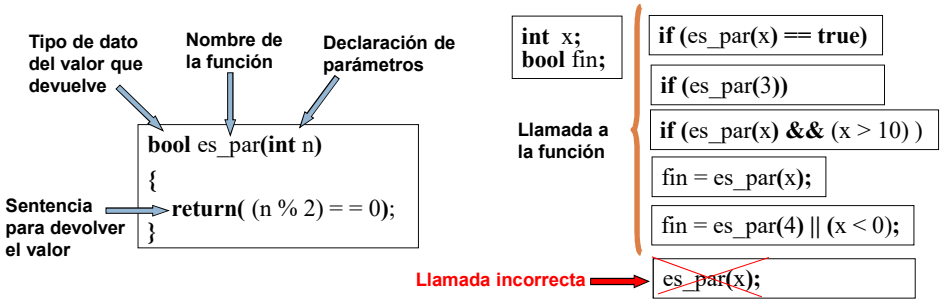
```
void rectangulo( int ancho, int alto,
                 int &area_rect, int &perim )
{
  area_rect = ancho * alto;
  perim = 2 * (ancho + alto);
}
```



Concepto de Función

15

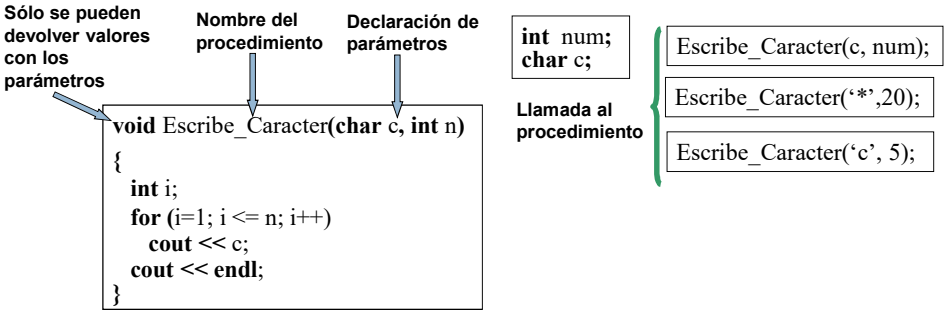
- Devuelve un valor que está asociado al nombre de la función
- Se suele definir con N parámetros (N ≥ 1)
- Sólo se debe utilizar paso de parámetros por valor



Concepto de Procedimiento

16

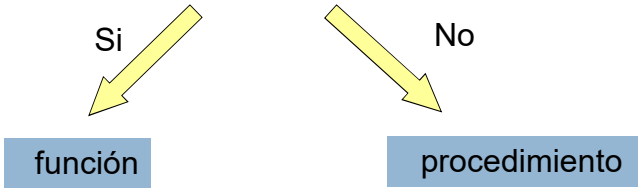
- Se puede definir con N parámetros (N ≥ 0)
- Se puede utilizar paso de parámetros por valor y/o por referencia
- Se invoca con una sentencia compuesta por su nombre y lista de parámetros actuales (la llamada es una sentencia por sí misma)



¿Qué uso: un procedimiento o una función?

17

¿El módulo tiene que **devolver un solo valor**?



Sobre la sentencia return

18

- 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*);

Concepto de ámbito de una variable

19

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

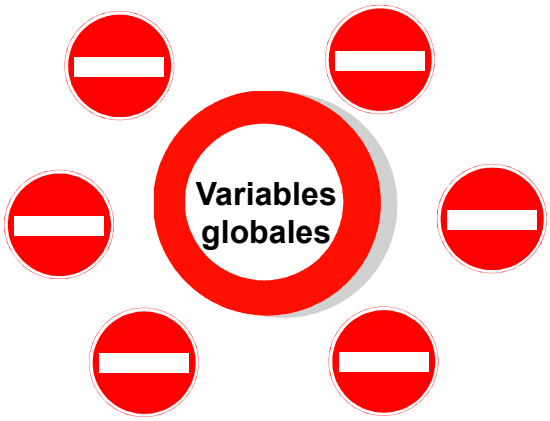
20

- 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

21

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



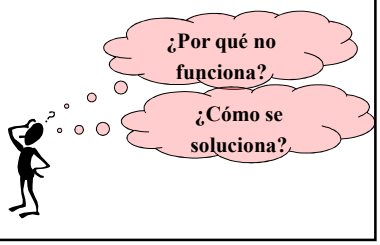
Efecto lateral

22

Cualquier comunicación de datos entre módulos al margen de los parámetros y la devolución de resultados se denomina efecto lateral

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

int resultado;  // declaración de variable global

int Num_Mayor(int n1, int n2);

main() {
  int  n1, n2;  // números introducidos por teclado   (datos de entrada)
  int  mayor; // el mayor de los dos números   (dato de salida)

  cout << "Introduce dos números :";
  cin >> n1 >> n2;
  resultado = n1 + n2;
  mayor = Num_Mayor(n1, n2);
  cout << "La suma de los dos números es: " << resultado;
  cout << "  y el mayor de ellos es:"  << mayor;
  cout <<  endl;
}
```

```
// función que devuelve el mayor de dos números
int Num_Mayor(int n1, int n2)
{
  if (n1 > n2)
    resultado = n1;
  else
    resultado = n2;

  return(resultado);
}
```



¿Qué tipo de programas debo ser capaz de hacer?

23

#directivas del preprocesador

Declaración de constantes

Declaración de procedimientos y funciones

main() {

Declaración de variables (de tipos simples)

Cuerpo principal

sentencias de control

llamadas a procedimientos y funciones

}

Definición de procedimientos y funciones

Ejemplo de programa

24

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

// Cambios de moneda a Euros
const float US_DOLAR_EURO =  1,4696;
const float LIBRA_ESTERLINA_EURO =  1,4696;

// Declaración de procedimientos y funciones
void Leer_Importe(float &cantidad, char &moneda);
float Cambio_En_Euros(float cantidad, char moneda);

main() {
  float cantidad;  // cantidad de dinero  (dato de entrada)
  char  moneda; // tipo de moneda  (dato de entrada)
  char  respuesta; // respuesta para continuar (dato de entrada)
  float  euros;  // cantidad en euros equivalente (dato de salida)

  do {
    Leer_Importe(cantidad, moneda);
    euros = Cambio_En_Euros(cantidad, moneda);
    cout << "El cambio en euros es:" << euros << endl;
    cout << "¿Desea introducir otro importe? (S/N) :";
    cin >> respuesta;
  } while ( (respuesta == 's') || (respuesta == 'S'));
}
```

```
// Leer de teclado un importe y el tipo de moneda
// validando los datos introducidos hasta que
// sean correctos
void Leer_Importe(float &cantidad, char &moneda)
{
  bool datos_correctos;
  do {
    cout << "Introduce cantidad de dinero y moneda (D/L):";
    cin << cantidad << moneda;
    datos_correctos = (cantidad > 0.0) &&
                      ( moneda == 'D' || moneda == 'L');
  } while ( ! datos_correctos);
}

// Devolver el cambio en euros equivalente al importe y moneda
// especificados
float Cambio_En_Euros(float cantidad, char moneda)
{
  switch (moneda) {
    case 'D' : euros = cantidad * US_DOLAR;
               break;
    case 'L' : euros = cantidad * LIBRA_ESTERLINA_EURO;
  }
  return (euros);
}
```

Cuando definas un módulo, recuerda incluir un **comentario** que explique **qué hace** el módulo

Bibliotecas del lenguaje C / C++

25

- 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++

26

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

27

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.

Bibliografía Recomendada

28

Fundamentos de Programación
Jesús Carretero, Félix García, y otros
Thomson-Paraninfo 2007. ISBN: 978-84-9732-550-9

Capítulo 7

Problemas Resueltos de Programación en Lenguaje C
Félix García, Alejandro Calderón, y otros
Thomson (2002) ISBN: 84-9732-102-2

Capítulo 5

Resolución de Problemas con C++
Walter Savitch
Pearson Addison Wesley 2007. ISBN: 978-970-26-0806-6

Capítulo 4