Programación 1

Tema 7

Desarrollo modular y descendente de programas

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Índice

- Programas dirigidos por menú
- Diseño modular
- Módulos de biblioteca en C++

Programa dirigido por menú

```
MENÚ DE OPERACIONES
===================
0 - Finalizar
1 - Calcular el número de cifras de un entero
2 - Sumar las cifras de un entero
3 - Extraer una cifra de un entero
4 - Calcular la imagen especular de un entero
5 - Comprobar si un entero es primo

Seleccione una operación [0-5]: 4
Escriba un número entero: 8802361
El número imagen especular del 8802361 es el 1632088

...
```

Programa dirigido por menú

```
...
MENÚ DE OPERACIONES
====================
0 - Finalizar
1 - Calcular el número de cifras de un entero
2 - Sumar las cifras de un entero
3 - Extraer una cifra de un entero
4 - Calcular la imagen especular de un entero
5 - Comprobar si un entero es primo

Seleccione una operación [0-5]: 5
Escriba un número entero: 103
El número 103 es primo
...
```

Programa dirigido por menú

```
...
MENÚ DE OPERACIONES
===================
0 - Finalizar
1 - Calcular el número de cifras de un entero
2 - Sumar las cifras de un entero
3 - Extraer una cifra de un entero
4 - Calcular la imagen especular de un entero
5 - Comprobar si un entero es primo

Seleccione una operación [0-5]: 7
Opción desconocida
...
```

Programa dirigido por menú

```
...
MENÚ DE OPERACIONES
====================
0 - Finalizar
1 - Calcular el número de cifras de un entero
2 - Sumar las cifras de un entero
3 - Extraer una cifra de un entero
4 - Calcular la imagen especular de un entero
5 - Comprobar si un entero es primo

Seleccione una operación [0-5]: 0
```

Estructura modular

- **Programas grandes**
  - Descomposición en **procedimientos y funciones**
    - Facilitan la descomposición de un problema y su solución,
    - Facilitan la escritura y el mantenimiento de un programa.
    - Incrementan su legibilidad.
    - Permiten reutilizar el código.
  - Descomposición en **módulos y bibliotecas**
    - Permiten desarrollo independiente (no necesariamente por un único programador).
    - Permiten reutilizar el código de forma más eficiente.

Estructura modular

- **Módulo de programa**
  - Contiene el código de la función principal del programa
- **Módulos de biblioteca**
  - Módulos adicionales en los que se puede dividir un programa y con los que puede contar

Estructura modular en C++

- **Módulo principal** obligatorio
  - Se define en él, al menos, la función `main`
  - Se almacena en un fichero con sufijo `.cc` o `.cpp`
- **Módulos de biblioteca**
  - Definen recursos puestos a disposición de otros módulos
    - Tipos de datos
    - Datos constantes [y variables]
    - Funciones

Estructura modular en C++

- **Módulos de biblioteca**
  - Constan de dos ficheros:
    - **Interfaz del módulo**
      - Declaraciones y especificaciones de los recursos visibles fuera del módulo
      - Se almacena en un **fichero de cabecera**, un fichero con sufijo `.hh` o `.hpp`
    - **Implementación del módulo**
      - Código de las funciones declaradas en la interfaz
      - Elementos auxiliares
      - Se almacena en un fichero con sufijo `.cc` o `.cpp`

Programa del ejemplo

- Diseño con una estructura modular aplicando una metodología descendente:
  - Módulo principal
    - Fichero `calculadora-main.cpp`
    - Gestiona la interacción con el usuario con un comportamiento iterativo:
      - Plantea el menú de opciones (operaciones disponibles).
      - Lee la opción seleccionada por el usuario.
      - Ejecuta la orden correspondiente a la opción elegida por el usuario.
  - Módulo de biblioteca `calculos`
    - Define siete funciones que realizan cálculos y análisis de propiedades de enteros.

Programa del ejemplo

- Módulo de biblioteca `calculos`
  - Define siete funciones que realizan cálculos y análisis de propiedades de enteros:
    - `unsigned numCifras(int n)`
    - `unsigned sumaCifras(int n)`
    - `unsigned cifra(int n, unsigned i)`
    - `int imagen(int n)`
    - `unsigned factorial(unsigned n)`
    - `bool esPrimo(unsigned n)`
    - `unsigned mcd(int a, int b)`
  - Compuesto por dos ficheros
    - Interfaz del módulo: fichero de cabecera `calculos.hpp`
    - Implementación del módulo: fichero `calculos.cpp`

Diseño descendente. Módulo principal. Primer nivel

```
/* Programa que solicita al usuario las coordenadas reales… */
int main() {
    unsigned operacion;
    pedirOrden(operacion);

    // Itera hasta que el valor de «operacion» sea igual a 0.
    while (operacion != 0) {
        ejecutarOrden(operacion);
        pedirOrden(operacion);
    }

    return 0;
}
```

Diseño descendente. Módulo principal. Segundo nivel

```
/*
 *  Pre:  ---
 *  Post: Presenta en la pantalla el menú de
 *        opciones disponibles, solicita al
 *        usuario que escriba el código de una
 *        de ellas y asigna a «operacion»
 *        la nueva respuesta del usuario.
 */
void pedirOrden(unsigned &operacion) {
    presentarMenu();
    cout << "Seleccione una operacion [0-5]: ";
    cin >> operacion;
}
```

Diseño descendente. Módulo principal. Tercer nivel

```
/*
 *  Pre:  ---
 *  Post: Presenta el menú de opciones disponibles
 */
void presentarMenu() {
     cout << endl;
     cout << "MENU DE OPERACIONES" << endl;
     cout << "===================" << endl;
     cout << "0 - Finalizar" << endl;
     cout << "1 - Calcular el numero de cifras de un entero" << endl;
     cout << "2 - Sumar las cifras de un entero" << endl;
     cout << "3 - Extraer una cifra de un entero" << endl;
     cout << "4 - Calcular la imagen especular de un entero" << endl;
     cout << "5 - Comprobar si un entero es primo" << endl << endl;
}
```

Diseño descendente. Módulo principal. Segundo nivel

```
/*
 * Pre:  ---
 * Post: Ejecuta las acciones asociadas a la orden cuyo código es
 *       «operacion».
 */
void ejecutarOrden(unsigned operacion) {
    if (operacion >= 1 && operacion <= 5) {
        // Se va a ejecutar una operación válida.
        // En primer lugar se pide al usuario que defina un número entero.
        cout << "Escriba un número entero: ";
        int numero;
        cin >> numero;

        if (operacion == 1) {
            ejecutarNumCifras(numero);
        } else if (operacion == 2) {...}
        ...
    } else {
        // El código de operación no es válido
        cout << "Opción desconocida" << endl;
    }
}
```

Diseño descendente. Módulo principal. Tercer nivel

```
/*
 * Pre:  ---
 * Post: Ejecuta la 1ª orden,
 *       informando del número de cifras
 *       de «numero».
 */
void ejecutarNumCifras(int numero) {
    cout << "El número " << numero
         << " tiene " << numCifras(numero)
         << " cifras." << endl;
}
```

Diseño descendente. Estructura del módulo principal

```
#include <iostream>
#include "calculos.hpp"
using namespace std;

void presentarMenu() {...}

void pedirOrden(unsigned &operacion) {...}

void ejecutarNumCifras(int numero) {...}

...

void ejecutarOrden(int operacion) {...}

int main() {...}
```

Diseño descendente.
4.º nivel. Módulo `calculos`. Interfaz

```
/*
 * Pre:  ---
 * Post: Devuelve el número de cifras de «n» cuando este
 *       número se escribe en base 10.
 */
unsigned numCifras(int n);

/*
 * Pre:  ---
 * Post: Devuelve la suma de las cifras de «n» cuando «n» se
 *       escribe en base 10.
 */
unsigned sumaCifras(int n);

...
```

Diseño descendente. 4.º nivel. Módulo `calculos`. Implementación

```
#include "calculos.hpp"

/*
 * Pre:  ---
 * Post: Devuelve el número de cifras de «n» cuando este número se
 *       escribe en base 10.
 */
unsigned numCifras(int n) {
   unsigned cuenta = 1; n = n / 10;
   while (n != 0) {
      cuenta++; n = n / 10;
   }
   return cuenta;
}

/*
 * Pre:  ---
 * Post: Devuelve la suma de las cifras de «n» cuando «n» se escribe
 *       en base 10.
 */
unsigned sumaCifras(int n) {
    ...
}
```

Diseño modular del programa

Módulo principal

calculadora-main.cpp

```
#include <iostream>
#include "calculos.hpp"

void presentarMenu() {…}
void ejecutarOrden(unsigned operacion) {…}
int main() {…}
```

Módulo `calculos`

calculos.hpp

```
unsigned numCifras(int n);
unsigned sumaCifras(int n);
unsigned cifra(int n, unsigned i);
int imagen(int n);
unsigned factorial(unsigned n);
bool esPrimo(unsigned n);
unsigned mcd(int a, int b);
```

calculos.cpp

```
#include "calculos.hpp"

unsigned numCifras(int n) {…}
unsigned sumaCifras(int n) {…}
unsigned cifra(int n, unsigned i) {…}
int imagen(int n) {…}
unsigned factorial(unsigned n) {…}
bool esPrimo(unsigned n) {…}
unsigned mcd(int a, int b) {…}
```

Compilación modular

calculos.hpp

```
unsigned numCifras(int n);
unsigned sumaCifras(int n);
unsigned cifra(int n, unsigned i);
int imagen(int n);
unsigned factorial(unsigned n);
bool esPrimo(unsigned n);
unsigned mcd(int a, int b);
```

calculadora-main.cpp

```
#include <iostream>
#include "calculos.hpp"

void presentarMenu() {…}
void ejecutarOrden(unsigned op) {…}
int main() {…}
```

calculos.cpp

```
#include "calculos.hpp"

unsigned numCifras(int n) {…}
unsigned sumaCifras(int n) {…}
unsigned cifra(int n, unsigned i) {…}
int imagen(int n) {…}
unsigned factorial(unsigned n) {…}
bool esPrimo(unsigned n) {…}
unsigned mcd(int a, int b) {…}
```

Compilación modular

Error

calculos.hpp

```
unsigned numCifras(int n);
unsigned sumaCifras(int n);
unsigned cifra(int n, unsigned i);
int imagen(int n);
unsigned factorial(unsigned n);
bool esPrimo(unsigned n);
unsigned mcd(int a, int b);
```

calculadora-main.cpp

```
#include <iostream>
#include "calculos.hpp"

void presentarMenu() {…}
void ejecutarOrden(unsigned op) {…}
int main() {…}
```

calculos.cpp

```
#include "calculos.hpp"

unsigned numCifras(int n) {…}
unsigned sumaCifras(int n) {…}
unsigned cifra(int n, unsigned i) {…}
int imagen(int n) {…}
unsigned factorial(unsigned n) {…}
bool esPrimo(unsigned n) {…}
unsigned mcd(int a, int b) {…}
```

```
> g++ -o calculadora.exe calculadora-main.cpp
undefined reference to `numCifras(int)'
undefined reference to `sumaCifras(int)'
undefined reference to `cifra(int, unsigned int)'
undefined reference to `imagen(int)'
undefined reference to `esPrimo(unsigned int)'
```

Compilación modular

Error

calculos.hpp

```
unsigned numCifras(int n);
unsigned sumaCifras(int n);
unsigned cifra(int n, unsigned i);
int imagen(int n);
unsigned factorial(unsigned n);
bool esPrimo(unsigned n);
unsigned mcd(int a, int b);
```

calculadora-main.cpp

```
#include <iostream>
#include "calculos.hpp"

void presentarMenu() {…}
void ejecutarOrden(unsigned op) {…}
int main() {…}
```

calculos.cpp

```
#include "calculos.hpp"

unsigned numCifras(int n) {…}
unsigned sumaCifras(int n) {…}
unsigned cifra(int n, unsigned i) {…}
int imagen(int n) {…}
unsigned factorial(unsigned n) {…}
bool esPrimo(unsigned n) {…}
unsigned mcd(int a, int b) {…}
```

```
> g++ -o calculadora.exe calculos.cpp
undefined reference to `main()'
```

Compilación modular

Una posibilidad… **lenta**


```
> g++ -o calculadora.exe calculos.cpp calculadora-main.cpp
```

Compilación modular

La forma habitual en C++: ficheros objeto

calculos.hpp

```
unsigned numCifras(int n);
unsigned sumaCifras(int n);
unsigned cifra(int n, unsigned i);
int imagen(int n);
unsigned factorial(unsigned n);
bool esPrimo(unsigned n);
unsigned mcd(int a, int b);
```

calculadora-main.cpp

```
#include <iostream>
#include "calculos.hpp"

void presentarMenu() {…}
void ejecutarOrden(unsigned op) {…}
int main() {…}
```

calculos.cpp

```
#include "calculos.hpp"

unsigned numCifras(int n) {…}
unsigned sumaCifras(int n) {…}
unsigned cifra(int n, unsigned i) {…}
int imagen(int n) {…}
unsigned factorial(unsigned n) {…}
bool esPrimo(unsigned n) {…}
unsigned mcd(int a, int b) {…}
```

calculos.o

```
numCifras: 10001011101…
sumaCifras: 1110100101…
cifra: 110100010010111…
imagen: 10001011101…
factorial: 01110100101…
esPrimo: 101101010111…
mcd: 10001011101…
```

```
> g++ -o calculos.o calculos.cpp
```

Compilación modular

La forma habitual en C++: ficheros objeto

iostream

```
...
#include <bits/c++config.h>
#include <ostream>
#include <istream>
…
istream cin;
ostream cout;
```

calculadora-main.cpp

```
#include <iostream>
#include "calculos.hpp"

void presentarMenu() {…}
void ejecutarOrden(unsigned op) {…}
int main() {…}
```

calculadora-main.o

presentarMenu: 10001011101…
ejecutarOrden: 01110100101…
main: 10110100010010111…
Falta código de: numCifras, sumaCifras, cifra, imagen, factorial, esPrimo, mcd, cin, cout, endl, flush, >>, <<

calculos.hpp

```
unsigned numCifras(int n);
unsigned sumaCifras(int n);
unsigned cifra(int n, unsigned i);
int imagen(int n);
unsigned factorial(unsigned n);
bool esPrimo(unsigned n);
unsigned mcd(int a, int b);
```

calculos.cpp

```
#include "calculos.hpp"

unsigned numCifras(int n) {…}
unsigned sumaCifras(int n) {…}
unsigned cifra(int n, unsigned i) {…}
int imagen(int n) {…}
unsigned factorial(unsigned n) {…}
bool esPrimo(unsigned n) {…}
unsigned mcd(int a, int b) {…}
```

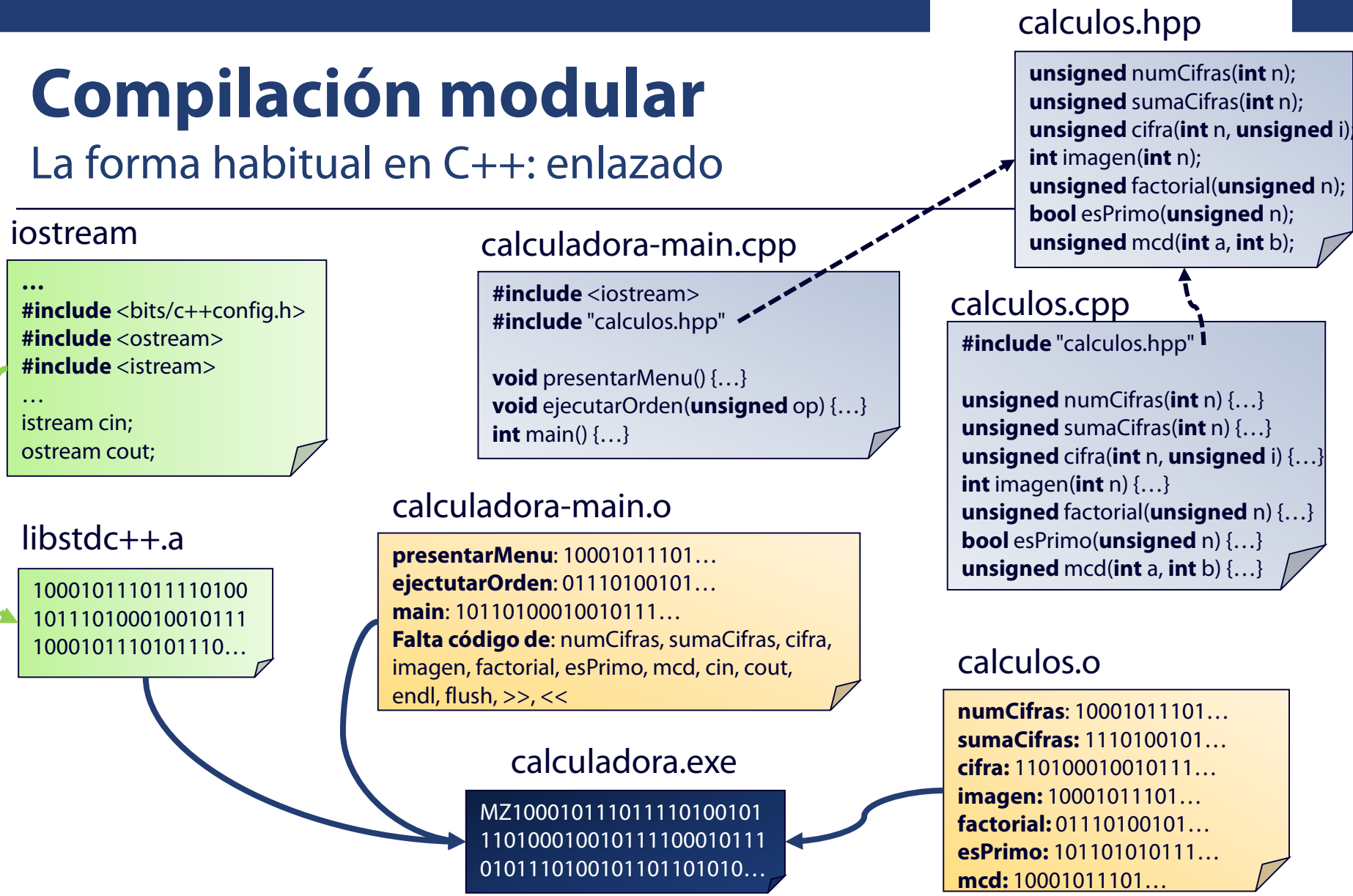
calculos.o

numCifras: 10001011101…
sumaCifras: 1110100101…
cifra: 110100010010111…
imagen: 10001011101…
factorial: 01110100101…
esPrimo: 101101010111…
mcd: 10001011101…

```
> g++ -o calculadora-main.o calculadora-main.cpp
```

Compilación modular

La forma habitual en C++: enlazado




```
> g++ -o calculadora.exe calculadora-main.o calculos.o
```

Compilación modular

La forma habitual en C++

```
g++ -o calculos.o -c calculos.cpp
g++ -o calculadora-main.o -c calculadora-main.cpp
g++ -o calculadora.exe calculadora-main.o calculos.o
```

Compilación modular
La forma habitual en C++

- Si ya hemos compilado todo una vez y modificamos solo un fichero (por ejemplo, «calculos.cpp»), no sería necesario compilar todo otra vez.
- Bastaría con compilar:
  - `g++ -o calculos.o -c calculos.cpp`
  - `g++ -o calculadora.exe calculadora-main.o calculos.o`
- No haría falta ejecutar:
  - `g++ -o calculadora-main.o -c calculadora-main.cpp`

Compilación modular
Ficheros makefile

```
calculadora.exe: calculadora-main.o calculos.o
    g++ -o calculadora.exe calculadora-main.o calculos.o

calculadora-main.o: calculadora-main.cpp calculos.hpp
    g++ -o calculadora-main.o -c calculadora-main.cpp

calculos.o: calculos.cpp calculos.hpp
    g++ -o calculos.o -c calculos.cpp
```

Compilación modular
Ficheros makefile

```
<fichero-makefile> ::= <regla> { <regla> }
<regla> ::= <objetivo> “:” <prerrequisitos> <fin-de-línea>
     { <receta> <fin-de-línea> }
<objetivo> ::= <nombre-fichero>
<prerrequisitos> ::= { <nombre-fichero> }
<receta> ::= <tabulador> <comando>
```

Compilación modular
Ficheros makefile

- El programa make ejecuta la primera regla del fichero makefile
- Para construir el fichero objetivo de una regla *R*:
  - Comprueba que los ficheros prerrequisito establecidos en la regla *R* existen y están actualizados
    - Si no existen o no están actualizados, recursivamente busca otra regla que los tenga como objetivo y la ejecuta.
    - Cuando los ficheros prerrequisito existen y están actualizados, si el fichero objetivo de la regla *R* no existe o no está actualizado (porque alguno de los ficheros prerrequisito es más reciente que el objetivo), ejecuta la receta correspondiente a la regla *R*.