Programación 1

Desarrollo de proyectos de programación

Desarrollo de programas C++ que trabajan con números romanos

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Objetivo del proyecto

- Desarrollar una serie de programas que trabajan con **números romanos**:
  - Un primer programa que emule el comportamiento de una **calculadora interactiva** que trabaje con números romanos
  - Un segundo programa que **procese** un fichero de texto en el cual hay escritas una secuencia de expresiones con números romanos
  - A los programas anteriores les pueden suceder **otros programas** que trabajen con números romanos

Metodología

- El diseño ha de ser modular y descendente
- Se ha de facilitar la reutilización de código
- El desarrollo del código debe venir precedido por una fase de análisis y una fase de diseño (arquitectura de los programas y especificación de las funciones de cada módulo): análisis → diseño → codificación

Comportamiento del primer programa

```
Expresión con romanos (FIN para acabar): MCDIX + CDLII
MCDIX + CDLII = MDCCCLXI

Expresión con romanos (FIN para acabar): M - IV
M - IV = CMXCVI

Expresión con romanos (FIN para acabar): XII x CIV
XII x CIV = MCCXLVIII

...
```

Comportamiento del primer programa

```
...

Expresión con romanos (FIN para acabar): MCC / LIV
MCC / LIV =  XXII

Expresión con romanos (FIN para acabar): FIN
```

Comportamiento ante situaciones anómalas (uso de minúsculas):

```
Expresión con romanos (FIN para acabar): mcdix + cdlii
MCDIX + CDLII = MDCCCLXI

Expresión con romanos (FIN para acabar): M – iv
M - IV = CMXCVI

Expresión con romanos (FIN para acabar): Xii x CIv
XII x CIV = MCCXLVIII

Expresión con romanos (FIN para acabar): ...
```

Comportamiento ante situaciones anómalas (desbordamiento en el resultado):

```
Expresión con romanos (FIN para acabar): MMMCII + MMXII
MMMCII + MMXII = 5114

Expresión con romanos (FIN para acabar): MMXII - MMMCII
MMXII - MMMCII = -1090

Expresión con romanos (FIN para acabar): ...
```

Comportamiento ante situaciones anómalas (error en planteamiento de operación):

```
Expresión con romanos (FIN para acabar): XX & CIX
*** ERROR: operador & desconocido

Expresión con romanos (FIN para acabar): ...
```

Comportamiento del segundo programa

```
Nombre del fichero a procesar: expresiones01.txt
Nombre del fichero de resultados: resultados01.txt

Se han procesado 15 expresiones:
  12 expresiones estaban bien formuladas
   3 expresiones estaban mal formuladas
```

Contenidos del fichero a procesar y del fichero de resultados

```
M - I
I - M
XII x LXIII
xii x lxIII
DCCCIV / XVIII
DCCCIV % XVIII
MIX + XVI
CLV + LXXII
CLV - LXXII
CLV x LXXII
CLV / LXXII
CLV * LXXII
MCCXXIV $ I
```

```
M - I = CMXCIX
I - M = -999
XII x LXIII = DCCLVI
XII x LXIII = DCCLVI
DCCCIV / XVIII = XLIV
*** ERROR: operador %
MIX + XVI = MXXV
CLV + LXXII = CCXXVII
CLV - LXXII = LXXXIII
CLV x LXXII = 11160
CLV / LXXII = II
*** ERROR: operador *
*** ERROR: operador $
```

Análisis del problema

- Pregunta previa:
  - ¿Qué es lo que más os preocupa hasta el momento?
  - ¿Cómo se escribe un número romano?
    - Lo analizaremos más tarde
- Fases
  - Fijaremos los requisitos de los dos programas
  - Los analizaremos y diseñaremos
  - Analizaremos el problema de la notación romana

Requisitos

- Requisitos del primer programa ("calculadora")
- Requisitos del segundo programa ("procesador")

Requisitos del primer programa

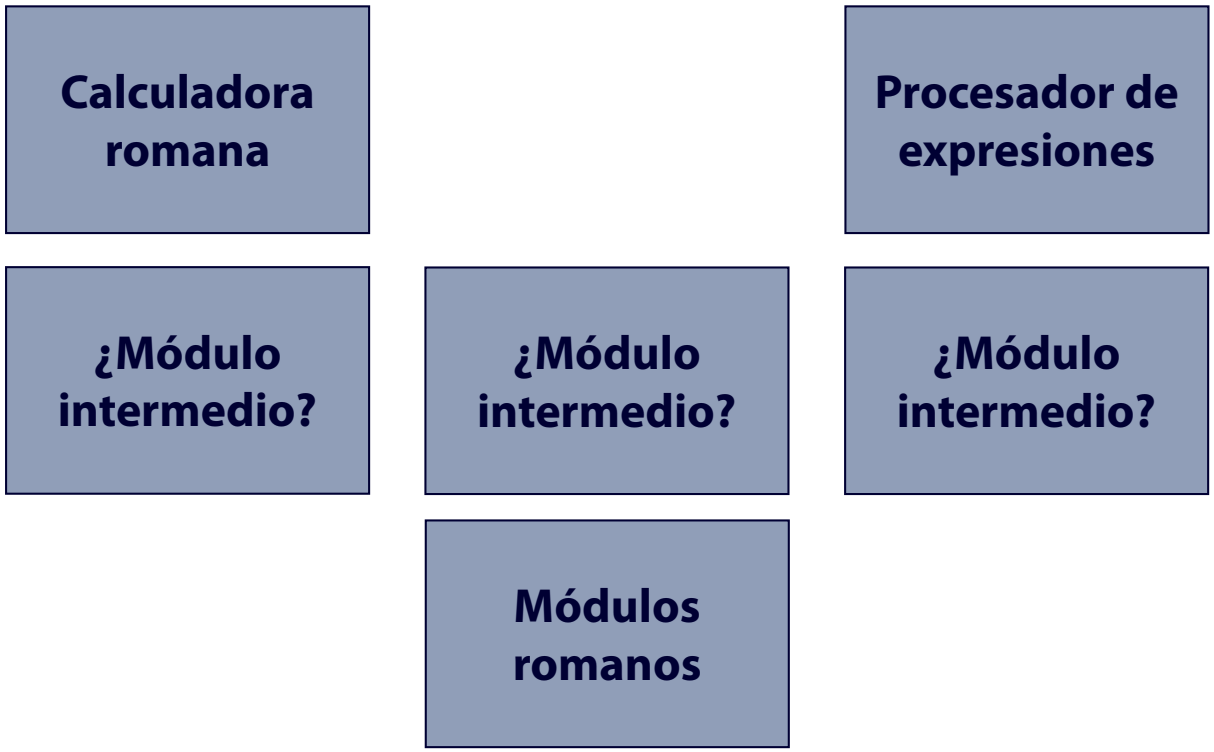
- Es interactivo
- Es iterativo
- Termina cuando se escribe “FIN”
- Pide y lee del teclado expresiones con operandos romanos
- Escribe en la pantalla expresiones con operandos romanos y su resultado
- Los operadores admitidos son «+» (suma), «-» (resta), «x» (multiplicación) y «/» (división)
- Los operadores están separados de los operandos al menos por un espacio en blanco
- Es irrelevante que los operandos o la palabra “FIN” estén escritos en mayúsculas o minúsculas
- Cuando un resultado no sea representable como número romano, se representará con números arábigos en base 10
- Cuando se plantee una expresión con un operador erróneo, se escribirá un mensaje de error en la pantalla y el programa continuará con normalidad

Requisitos del segundo programa

- Es interactivo
- Pide y lee del teclado el nombre de un fichero de texto que contiene expresiones romanas, una por línea
- Pide y lee del teclado el nombre de un fichero de texto en el que dejar los resultados
- Lee del primer fichero de texto expresiones con operandos romanos
- Escribe en el segundo fichero de texto expresiones con operandos romanos y su resultado
- Los operadores binarios admitidos son «+» (suma), «-» (resta), «x» (multiplicación) y «/» (división)
- Los operadores están separados de los operandos al menos por un espacio en blanco
- Es irrelevante que los operandos estén escritos en mayúsculas o minúsculas
- Cuando un resultado no sea representable como número romano, se representará en números arábigos en base 10
- Cuando se lea del primer fichero de texto una expresión con un operador erróneo, se escribirá un mensaje de error en el segundo fichero y el programa continuará con normalidad
- Al terminar, informa del número de expresiones que se han procesado, indicando asimismo cuántas estaban bien formuladas y cuántas mal formuladas.

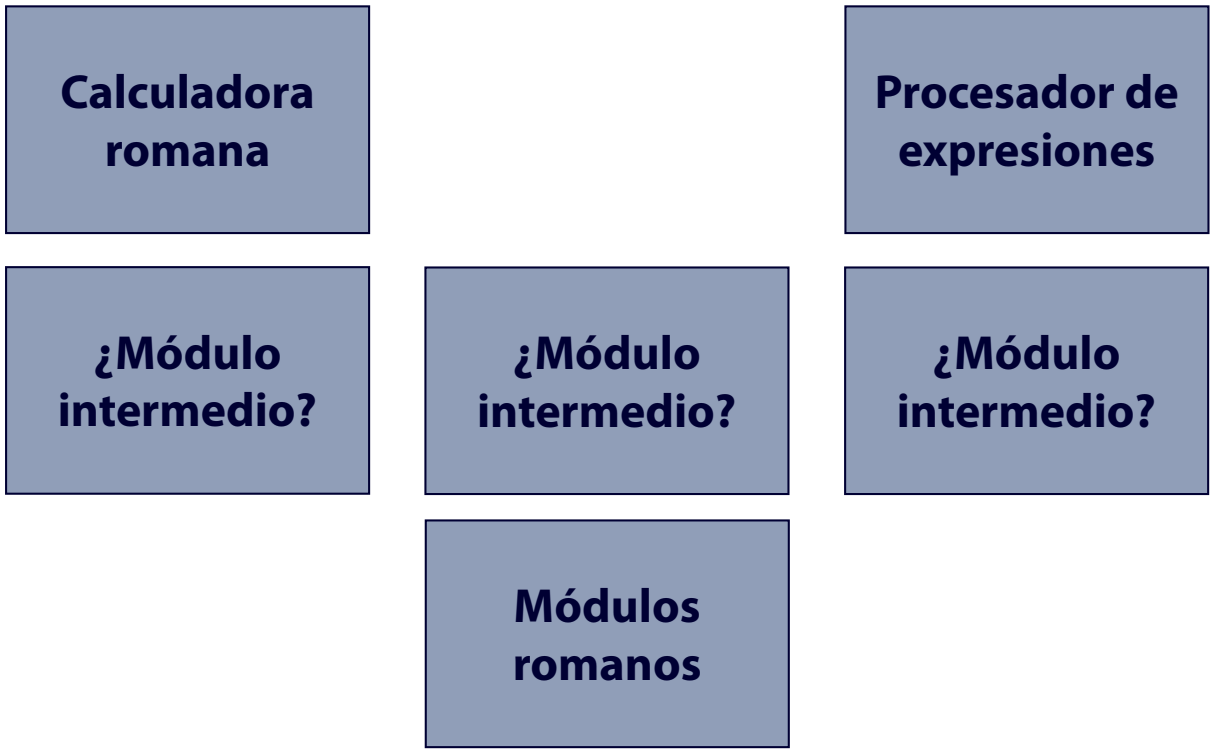
Diseño de los programas

- Arquitectura de los programas:
  - ¿Qué módulos los han de integrar?



Diseño de los programas

- Arquitectura de los programas:
  - ¿Qué módulos los han de integrar?



Módulo «Calculadora romana»

- Función `main` que:
  - Repite hasta que se escribe `FIN`
    - Pide una expresión con romanos
    - Lee una expresión con romanos del teclado
    - Escribe la expresión y su resultado en la pantalla
  - Hasta que la expresión leída es `FIN`

Módulo «Procesador de expresiones»

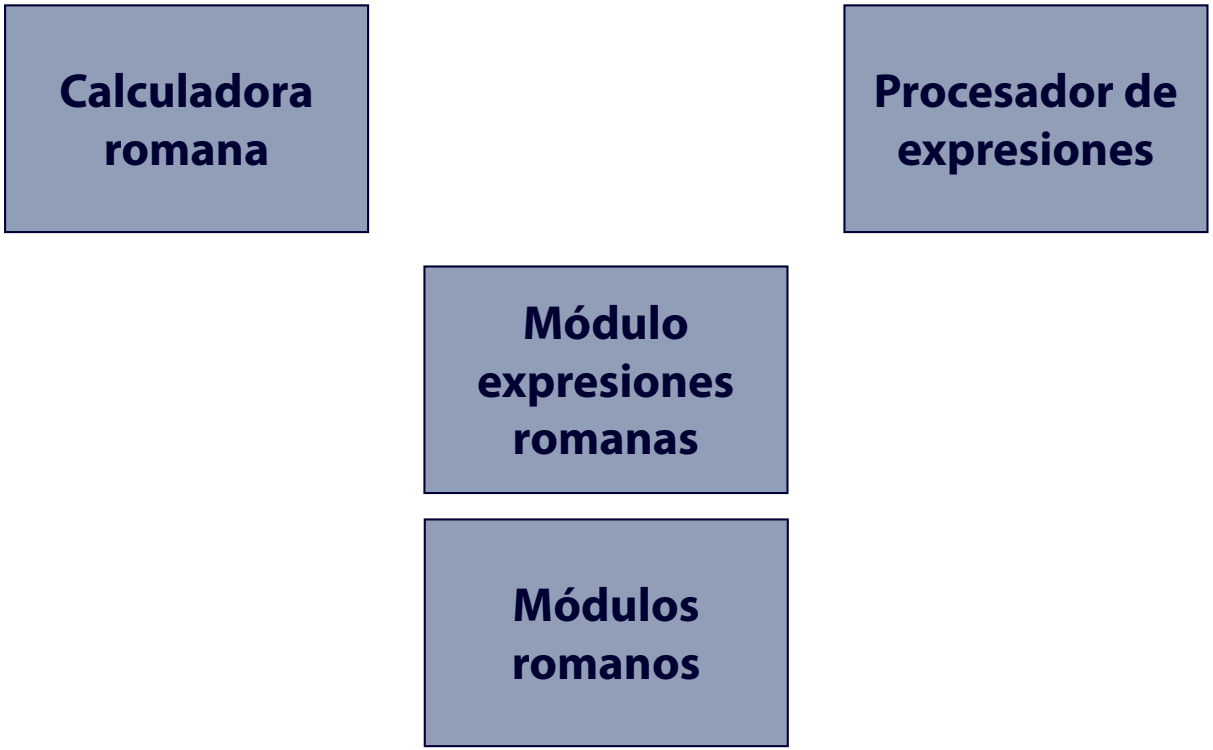
- Función `main` que:
  - Pide el nombre de dos ficheros de texto
  - Repite hasta que se acaba el fichero:
    - Lee una expresión con romanos del primer fichero
    - Escribe la expresión y su resultado en el segundo fichero
  - Escribe en pantalla un resumen de las operaciones realizadas

Similitudes módulos «Calculadora romana» y «Procesador de expresiones»

- Lee del teclado una expresión con romanos
- Escribe en la pantalla la expresión romana y su resultado
- Lee de un fichero una expresión con romanos
- Escribe en otro fichero la expresión romana y su resultado

Diseño de los programas

- Arquitectura de los programas:
  - Una posible solución



Módulo «Expresiones romanas»

- Función `bool procesar(istream& entrada, ostream& salida, bool& correcta)` que:
  - Intenta leer de «`entrada`» una expresión con números romanos
  - Si la puede leer y es correcta:
    - Escribe en «salida» la expresión y el resultado
    - Asigna a «correcta» el valor «true»
    - Devuelve «true»
  - Si la puede leer y no es correcta:
    - Escribe en «salida» un mensaje de error
    - Asigna a «correcta» el valor «false»
    - Devuelve «true»
  - Si no la puede leer o si la expresión es la orden "FIN“
    - Devuelve «false»

Módulo «Expresiones romanas»

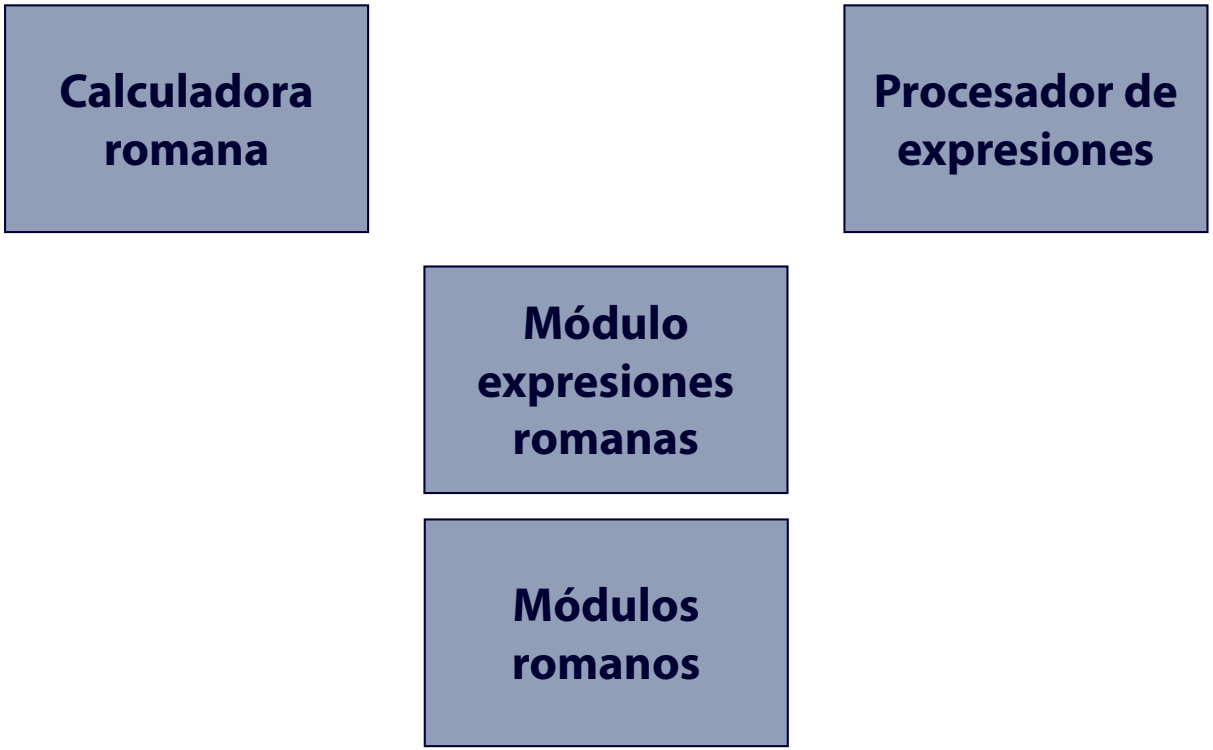
- **Leer de «entrada» una expresión con números romanos:**
  - Lee el primer operando romano y calcular su valor entero
  - Lee el operador
    - Si el operador es correcto:
      - Lee el segundo operando romano y calcular su valor entero
      - Escribe la expresión y el resultado en el segundo fichero
      - Asigna a «correcta» el valor «true»
    - Si el operador no es correcto
      - Asigna a «correcta» el valor «false»

Módulo «Expresiones romanas»

- **Escribir en «salida» la expresión y el resultado:**
  - Escribir el primero operando en notación romana
  - Escribir el operador
  - Escribir el segundo operando en notación romana
  - Escribir el signo «=»
  - Calcular el resultado
  - Escribir el resultado en notación romana

Diseño de los programas

- Arquitectura de los programas:
  - Una posible solución



Módulo «Romanos»

- ¿Representación de un número romano?
- ¿Funciones para trabajar con números romanos?

Módulo Romanos

- Una posible solución
  - Representación: cadenas de caracteres acabadas en ‘\0’
- ¿Funciones para trabajar con números romanos?
  - Calcular el valor entero de un número romano
  - Convertir un número entero en número romano

Módulo Romanos

- Representación
  - `const int MAX_LONG_ROMANO = 17;`
  - `char romanoEjemplo[MAX_LONG_ROMANO];`
- Funciones
  - `int valorDeRomano(const char romano[]);`
  - `void convertirARomano(const int valor, char romano[]);`

Módulo Romanos. Cabecera

```
const int MAX_LONG_ROMANO = 17;

/*
 * Pre:  «romano» corresponde a un número
 *       romano correcto.
 * Post: Ha devuelto el valor entero
 *       equivalente al número romano
 *       representado por «romano».
 */
int valorDeRomano(const char romano[]);
```

Módulo Romanos. Cabecera

```
/*
 * Pre:  ---
 * Post: Ha asignado a «romano» la secuencia de
 *       caracteres que definen un número romano
 *       equivalente a «valor». Si «valor» no es
 *       representable por tener un valor absoluto
 *       mayor o igual que 4000, ha asignado a
 *       «romano» el valor "***".
 */
void convertirARomano(const int valor,
                      char romano[]);
```

Números romanos

- Entero en base 10
  - 1
  - 4
  - 9
  - 90
  - 99
  - 149
  - 1008
  - 2854
  - 3709
  - 3999
  - 4120
- Entero en notación romana
  - I
  - IV
  - IX
  - XC
  - XCIX
  - CXLIX
  - MVIII
  - MMDCCCLIV
  - MMMDCCIX
  - MMMCMXCIX
  - ???

Números romanos. Reglas

I	1
V	5
X	10
L	50
C	100
D	500
M	1000

- **Regla de adición:** Las cifras romanas se escriben generalmente en orden de valores descendente
  - Un símbolo menor o igual a la derecha de otro mayor, suma su valor a este
- **Regla de sustracción**: un símbolo menor a la izquierda de otro mayor, resta su valor a este
  - I va solo a la izquierda de V o X
  - X va solo a la izquierda de L o C
  - C va solo a la izquierda de D o M
  - No se pueden anteponer los símbolos V, L y D
- **Principio de economía en la repetición**: se pueden repetir hasta tres veces los símbolos I, X, C, M y sus valores se suman
  - La aplicación de este principio es opcional y da lugar a la escritura de números romanos con el menor número posible de cifras

Números romanos

Número	Unidades	Decenas	Centenas	Millares
0				
1	I	X	C	M
2	II	XX	CC	MM
3	III	XXX	CCC	MMM
4	IV	XL	CD	
5	V	L	D	
6	VI	LX	DC	
7	VII	LXX	DCC	
8	VIII	LXXX	DCCC	
9	IX	XC	CM	

Números romanos

Cómo se escribe en romanos el 2086:

Número	Unidades	Decenas	Centenas	Millares
0				
1	I	X	C	M
2	II	XX	CC	MM
3	III	XXX	CCC	MMM
4	IV	XL	CD	
5	V	L	D	
6	VI	LX	DC	
7	VII	LXX	DCC	
8	VIII	LXXX	DCCC	
9	IX	XC	CM	

Números romanos

Cómo se escribe en romanos el 2086:

Número	Unidades	Decenas	Centenas	Millares
0				
1	I	X	C	M
2	II	XX	CC	MM
3	III	XXX	CCC	MMM
4	IV	XL	CD	
5	V	L	D	
6	VI	LX	DC	
7	VII	LXX	DCC	
8	VIII	LXXX	DCCC	
9	IX	XC	CM	

Números romanos

I	1
V	5
X	10
L	50
C	100
D	500
M	1000

- ¿Cuál es el entero representado por MCMIV?
  - **Regla de adición:** Un símbolo menor o igual a la derecha de otro mayor, suma su valor a este
  - **Regla de sustracción**: un símbolo menor a la izquierda de otro mayor, resta su valor a este
- MCMIV = M – C + M – I + V =
  = 1000 – 100 + 1000 – 1 + 5 = 1904

Números romanos. Sintaxis

<romano> ::= …

Números romanos. Sintaxis

```
<romano> ::= <millares> <centenas> <decenas> <unidades>

<millares> ::=  { "M" }
<centenas> ::=  { "C" } | "CD" | ("D" { "C" }) | "CM"
<decenas>  ::=  { "X" } | "XL" | ("L" { "X" }) | "XC"
<unidades> ::=  { "I" } | "IV" | ("V" { "I" }) | "IX"
```