



Tema 1

Diseño de
algoritmos y
programas

C++

Elementos básicos
Identificadores
Constantes
Variables
Tipos de datos
Expresiones
Entrada / salida
Control de flujo
Vectores y matrices
Cadenas de
caracteres en C
Registros
Tipos enumerados
Funciones
Estructura de un
programa

Metodología

Tema 1: Introducción Programación 2

Curso 2013-2014

Notas



Tema 1

Diseño de
algoritmos y
programas

C++

Elementos básicos
Identificadores
Constantes
Variables
Tipos de datos
Expresiones
Entrada / salida
Control de flujo
Vectores y matrices
Cadenas de
caracteres en C
Registros
Tipos enumerados
Funciones
Estructura de un
programa

Metodología

Índice

- 1 Diseño de algoritmos y programas
- 2 Conocimientos básicos de C++
- 3 Metodología recomendada

Notas

Cómo se hace un programa ...

- ❶ Estudio del problema y de las posibles soluciones
- ❷ Diseño del algoritmo [en papel](#)
- ❸ Escritura del programa [en el ordenador](#)
- ❹ Compilación del programa y corrección de errores
- ❺ Ejecución del programa
- ❻ ... y prueba de todos los casos posibles (o casi)

El proceso de escribir, compilar, ejecutar y probar debería ser iterativo, haciendo pruebas de funciones o módulos por separado del programa.

Notas

Elementos básicos del lenguaje

- Identificadores: nombres de variables, funciones, constantes, ...
- Palabras reservadas: `if`, `while`, ...
- Símbolos: `{ }` `()` `[]` `;`
- Constantes: `123` `12.3` `'a'`
- Tipos de datos:

TIPO	TAMAÑO (BITS)
<code>int</code>	32
<code>char</code>	8
<code>float</code>	32
<code>double</code>	64
<code>bool</code>	8
<code>void</code>	??

Notas

Identificadores

- Nombres de constantes, variables, funciones: deben ser *significativos* siempre que sea posible, El nombre debe indicar para qué se utiliza. Por ejemplo:

```
int numeroAlumnos = 0;
void VisualizarAlumnos(...)
```

Malos ejemplos:

```
const int KOCHO=8;
int p,q,r,a,b;
int contador1,contador2; // mejor int i,j;
```

- En C++ (y en muchos otros lenguajes) hay *palabras reservadas*, que no se pueden utilizar como nombres definidos por el usuario:

```
int friend long auto public union
```

Notas

Constantes

- Constantes en un programa:

TIPO	CONSTANTES
int	123 007 1010101
float/double	123. .123 1e1 1.231E-12
char	'a' '1' ';' '\n' '\0' '\\'
cadena (char [])	" " "hola" "doble: \"
bool	true false

- Constantes explícitas (declaradas por el programador):

```
const int MAXALUMNOS=600;
const double PI=3.141592;
const char DESPEDIDA[] = "ADIOS";
```

- ¿Qué constantes debo declarar como constante explícita? **Aquellas que podrían cambiar en futuras versiones del programa (por ejemplo, el tamaño del tablero en el juego del sudoku podría cambiar a 4, 16, 25, ...).**

Notas

Variables (1/3)

- Siempre que se declare una variable se debe inicializar, excepto cuando lo primero que se va a hacer después de declarar la variable es asignarle valor

```
int numeroProfesores=0;
int i;

for (i=0;i<MAXPERSONAS;i++) ...
```

- Las variables se declaran siempre dentro de una función (o dentro de un bloque entre llaves contenido en una función), si se declaran fuera de las funciones son *variables globales*. **En general, se recomienda no utilizar variables globales (son peligrosas). En P2 está prohibido**

Notas

Variables (2/3)

Variables globales:

```
#include <iostream>
using namespace std;
int contador=10;

void CuentaAtras(void)
{
    while (contador > 0)
    {
        cout << contador << " ";
        contador--;
    }
    cout << endl;
}

int main()
{
    CuentaAtras();
    ...
    CuentaAtras();    // Aqui no imprime nada
}
```

Notas

Tipos de datos (2/2)

Declaración de tipos:

- En C y C++ (y en otros lenguajes) se pueden definir tipos nuevos:
typedef *(el resto como una declaración de variables)*

```
typedef int entero;
entero i, j;

typedef bool logico, booleano;
// logico y booleano son bool
```

- Es posible declarar un vector como un tipo

```
typedef char cadena[MAXCADENA];
// cadena es un vector de char
```

pero no es recomendable.

- Además, en C++ los nombres que aparecen después de struct (y class, union) son tipos.

Notas

Expresiones (1/4)

Expresiones aritméticas:

- operandos enteros (int) y reales (float y double), operadores aritméticos (+ - * /).
- Si aparece un operando de tipo char o bool, se convierte a entero implícitamente (por ejemplo, 2+'a' vale 99)
- División entera: si los dos operandos son enteros, la división es entera. Si queremos que el resultado sea real, hay que hacer una conversión explícita (cast):
(float)7 / 2 vale 3.5
- Operador % : resto de la división entera (por ejemplo, 30 % 7 vale 2)
- Precedencia de operadores: 2+3*4 es 2+(3*4), no (2+3)*4 . En caso de duda poner paréntesis

Notas

Expresiones (4/4)

Expresiones lógicas:

- Operadores: ! (negación), && (y lógico), || (o lógico)
- Precedencia: (a || b && c) es (a || (b && c))
- Evaluación en cortocircuito:
 - Si el operando izquierdo de && es falso, el operando derecho no se evalúa (false && loquesea es siempre false)
 - Si el operando izquierdo de || es cierto, el operando derecho no se evalúa (true || loquesea es siempre true)

Notas

Entrada / salida

- Salida por pantalla:

```
cout << ... ;
```

- Salida de error:

```
cerr << ... ;
```

- Entrada:

```
cin >> ... ;
```

¿Cómo funciona? Lee blancos y tabuladores hasta leer la variable que se le indica, y deja el puntero de lectura justo después. [Puede dar problemas al leer cadenas después de otros datos \(más detalles en el tema 2\).](#)

Notas

Control de flujo (3/3)

- switch: la expresión debe ser entera (dará error si es float o double)

```
switch (...)
{
    case ...:
        ...
        break;
    case ...:
        ...
        break;
    default:
        ...
        break;
}
```

En P2 no se puede utilizar break para salir de un bucle.
Se debe utilizar variables booleanas.

Notas

Vectores y matrices (1/2)

- Tamaño fijo: usando constantes para el tamaño

```
int vectorAlumnos[MAXALUMNOS];
char tablero[MAXTABLERO][MAXTABLERO];
```

- Tamaño variable

```
int tamvector, vector[tamvector];
int nfilas, matriz[nfilas][MAXCOL];
```

Error grave: tamvector y nfilas están sin inicializar; podrían valer 100, 1211311 o 0, y provocar fallos de segmentación.

```
int tamvector, nfilas; cin >> tamvector >> nfilas;
int vector[tamvector], matriz[nfilas][MAXCOL];
```

Notas

Vectores y matrices (2/2)

- Asignación y acceso a valores:

```
vector[0] = 7;  
...  
vector[tamvector-1] = vector[tamvector-2]+1;
```

`vector[tamvector]` está fuera del vector → posible fallo de segmentación.

- Es posible utilizar filas de matrices (bidimensionales) como vectores:

```
LeeVector(matriz[j]);
```

Siempre se debe comprobar que no se sobrepasan los límites (de 0 a tamvector-1)

```
for (i=0; i<tamvector && ... ; i++) ...  
if (j>=0 && j<nfilas && k>=0 && k<MAXCOL) ...
```

Notas

Cadenas de caracteres en C

- Son vectores que contienen una secuencia de caracteres terminada en el carácter nulo '\0'

```
const char cadena[]="hola";
```

"hola"

h	o	l	a	\0
---	---	---	---	----

- Si lo declaramos como una variable **debemos** especificar su tamaño:

```
const int tCADENA = 10;  
char cadena[tCADENA];
```

- Importante:** "a" es una cadena, 'a' es un carácter

Notas



Tema 1

Diseño de algoritmos y programas

C++

Elementos básicos
Identificadores
Constantes
Variables
Tipos de datos
Expresiones
Entrada / salida
Control de flujo
Vectores y matrices

Cadenas de caracteres en C

Registros
Tipos enumerados
Funciones
Estructura de un programa

Metodología

Vectores STL

- Standard Template Library (STL) es una biblioteca de C++ de clases dinámicas, algoritmos e iteradores
- Contiene entre otras la clase vector: <http://www.cplusplus.com/reference/vector/vector/>

```
#include <vector>

vector<int> v; // Declara un vector de enteros
vector<int> v2(3); // Declara un vector de 3 enteros
v.resize(4); // Cambia dinamicamente su tamaño

v.push_back(12); // Anyade un valor al final del vector

// OJO: Esto deberia hacerse con iteradores, pero en P2
// se permite acceder asi en lugar de usar punteros
for (unsigned int i=0; i<v.size(); i++)
    v[i]=23; // Asignacion
```

- Los vectores tienen funciones para ordenación, borrado, etc.

Notas



Tema 1

Diseño de algoritmos y programas

C++

Elementos básicos
Identificadores
Constantes
Variables
Tipos de datos
Expresiones
Entrada / salida
Control de flujo
Vectores y matrices

Cadenas de caracteres en C

Registros
Tipos enumerados
Funciones
Estructura de un programa

Metodología

Registros

```
struct Alumno {           // 'Alumno' solo es un tipo
    int dni;               // en C++
    double nota;
};
```

Forma recomendada en P2:

```
typedef struct {           // 'Alumno' es un tipo en C
    int dni;               // y C++
    double nota;
} Alumno;
```

Uso de registros:

```
Alumno a,b;
...
a.dni = 123133; // asignacion a un campo
b = a;          // asignacion de un registro
               // (ojo, copia bit a bit)
```

Notas



Tema 1

Diseño de algoritmos y programas

C++

Elementos básicos
Identificadores
Constantes
Variables
Tipos de datos
Expresiones
Entrada / salida
Control de flujo
Vectores y matrices
Cadenas de caracteres en C
Registros
Tipos enumerados
Funciones
Estructura de un programa

Metodología

Tipos enumerados

- Los tipos enumerados pueden declararse con un conjunto de posibles valores, conocidos como enumeradores.
- Las variables de los tipos enumerados pueden tomar cualquier valor de estos enumeradores.

```
enum colors_e {black, blue, green, red};

colors_e mycolor;

mycolor = blue;
if (mycolor == green) mycolor = red;
```

- Los valores de los tipos enumerados se convierten implícitamente en enteros (`int`), y viceversa.

Notas



Tema 1

Diseño de algoritmos y programas

C++

Elementos básicos
Identificadores
Constantes
Variables
Tipos de datos
Expresiones
Entrada / salida
Control de flujo
Vectores y matrices
Cadenas de caracteres en C
Registros
Tipos enumerados
Funciones
Estructura de un programa

Metodología

Funciones (1/4)

- Una función es un conjunto de líneas de código que realizan una tarea y puede devolver un valor.
- Puede recibir parámetros.

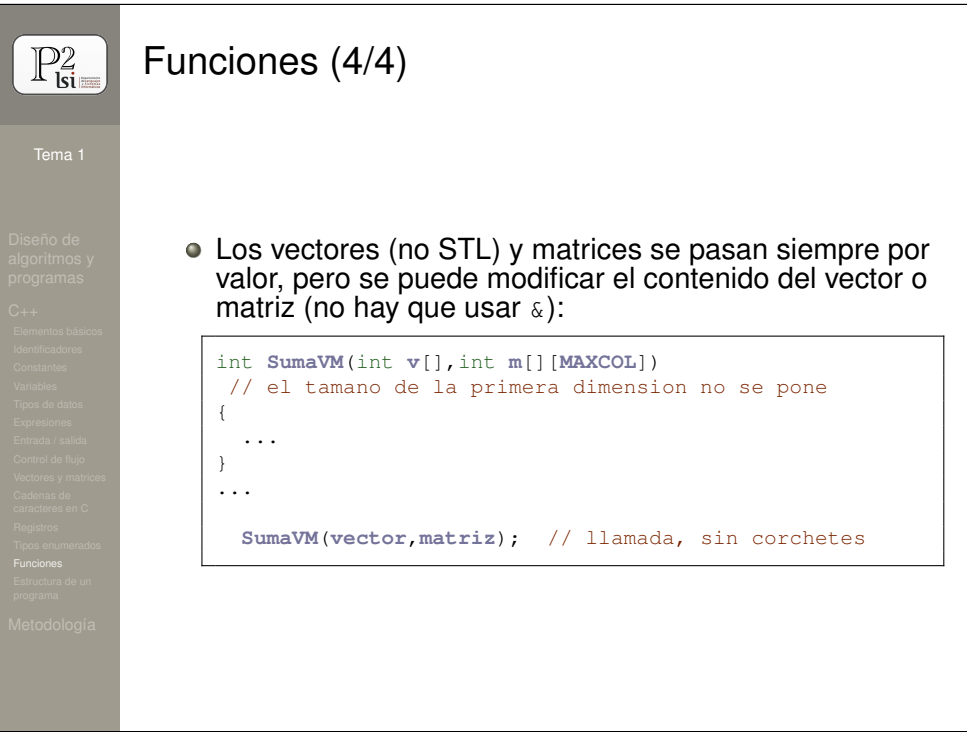
```
tipo_retorno nombre(parametro 1, parametro 2, ...)
{
    tipo_retorno ret;

    instruccion 1;
    instruccion 2;
    ...

    return ret;
}
```

- Una función no debería tener mucho código
- Si tengo que hacer copy & paste es porque necesito una función.
- **IMPORTANTE: En P2 no se puede usar más de un `return` en el cuerpo de una función.**

Notas



Funciones (4/4)

Tema 1

Diseño de algoritmos y programas

C++

Elementos básicos

Identificadores

Constantes

Variables

Tipos de datos

Expresiones

Entrada / salida

Control de flujo

Vectores y matrices

Cadenas de caracteres en C

Registros

Tipos enumerados

Funciones

Estructura de un programa

Metodología

- Los vectores (no STL) y matrices se pasan siempre por valor, pero se puede modificar el contenido del vector o matriz (no hay que usar &):

```
int SumaVM(int v[],int m[][MAXCOL])  
    // el tamaño de la primera dimension no se pone  
{  
    ...  
}  
...  
  
SumaVM(vector,matriz); // llamada, sin corchetes
```

- ## Funciones (4/4)
- ### Tema 1
- #### Diseño de algoritmos y programas
- #### C++
- #### Elementos básicos
- #### Identificadores
- #### Constantes
- #### Variables
- #### Tipos de datos
- #### Expresiones
- #### Entrada / salida
- #### Control de flujo
- #### Vectores y matrices
- #### Cadenas de caracteres en C
- #### Registros
- #### Tipos enumerados
- #### Funciones
- #### Estructura de un programa
- #### Metodología
- Los vectores (no STL) y matrices se pasan siempre por valor, pero se puede modificar el contenido del vector o matriz (no hay que usar &):
- ```
int SumaVM(int v[],int m[][MAXCOL])
 // el tamaño de la primera dimension no se pone
{
 ...
}
...

SumaVM(vector,matriz); // llamada, sin corchetes
```

## Funciones (4/4)

### Tema 1

#### Diseño de algoritmos y programas

#### C++

#### Elementos básicos

#### Identificadores

#### Constantes

#### Variables

#### Tipos de datos

#### Expresiones

#### Entrada / salida

#### Control de flujo

#### Vectores y matrices

#### Cadenas de caracteres en C

#### Registros

#### Tipos enumerados

#### Funciones

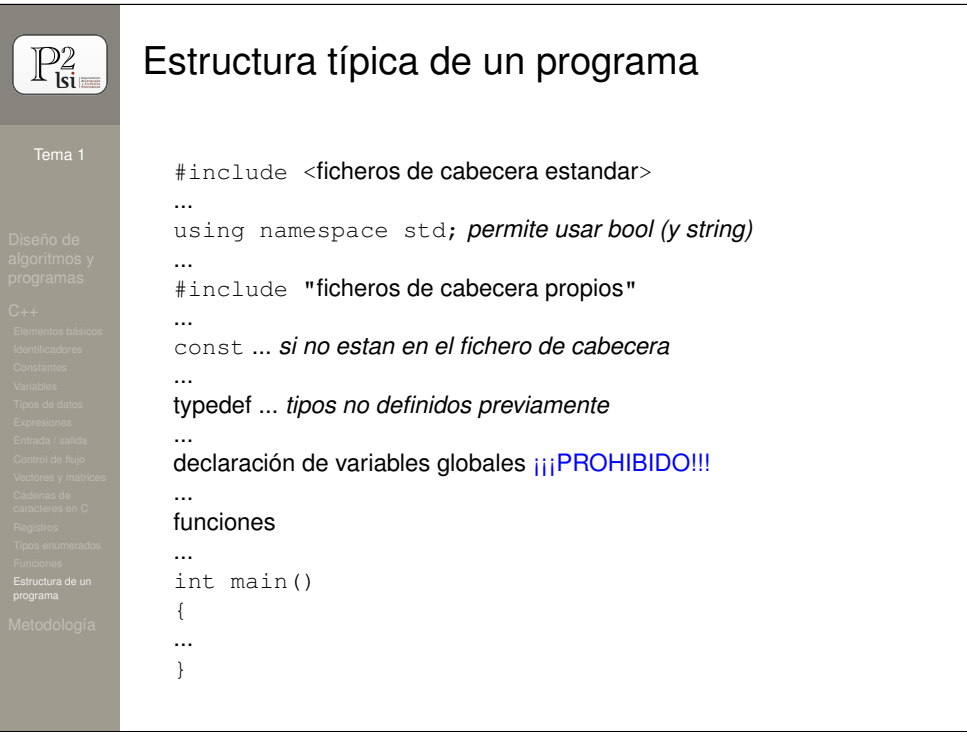
#### Estructura de un programa


#### Metodología

- Los vectores (no STL) y matrices se pasan siempre por valor, pero se puede modificar el contenido del vector o matriz (no hay que usar &):

```
int SumaVM(int v[],int m[][MAXCOL])
 // el tamaño de la primera dimension no se pone
{
 ...
}
...

SumaVM(vector,matriz); // llamada, sin corchetes
```

[illegible]



# Estructura típica de un programa

Tema 1

Diseño de  
algoritmos y  
programas

C++

Elementos básicos

Identificadores

Constantes

Variables

Tipos de datos

Expresiones

Entrada / salida

Control de flujo

Vectores y matrices

Cadenas de  
caracteres en C

Registros


Tipos enumerados

Funciones

Estructura de un  
programa

Metodología

```
#include <ficheros de cabecera estandar>
...
using namespace std; permite usar bool (y string)
...
#include "ficheros de cabecera propios"
...
const ... si no estan en el fichero de cabecera
...
typedef ... tipos no definidos previamente
...
declaración de variables globales !!!PROHIBIDO!!!
...
funciones
...
int main()
{
...
}
```



# Estructura típica de un programa

Tema 1

Diseño de  
algoritmos y  
programas

C++

Elementos básicos

Identificadores

Constantes

Variables

Tipos de datos

Expresiones

Entrada / salida

Control de flujo

Vectores y matrices

Cadenas de  
caracteres en C

Registros

Tipos enumerados

Funciones

Estructura de un  
programa

Metodología

```
#include <ficheros de cabecera estandar>
...
using namespace std; permite usar bool (y string)
...
#include "ficheros de cabecera propios"
...
const ... si no estan en el fichero de cabecera
...
typedef ... tipos no definidos previamente
...
declaración de variables globales !!!PROHIBIDO!!!
...
funciones
...
int main()
{
...
}
```

[illegible]



## Tema 1

Diseño de algoritmos y programas

C++

Elementos básicos

Identificadores

Constantes

Variables

Tipos de datos

Expresiones

Entrada / salida

Control de flujo

Vectores y matrices

Cadenas de caracteres en C

Registros

Tipos enumerados

Funciones

Estructura de un programa

Metodología

# Metodología recomendada para programar

- Estudio del problema y de la solución, y diseño del algoritmo en papel
- Diseñar el programa intentando hacer muchas funciones, con poco código (no sobrepasar las 30 líneas por función), y dejar poco código en el `main`, pero sin exagerar:

```
int main()
{
 Principal(); // incorrecto
}
```

- Evitar código repetido utilizando adecuadamente las funciones
- Compilar y probar las funciones por separado, no esperar a tener todo el programa para empezar a compilar y probar.

## Notas

---

---

---

---

---

---

---

---

---

---

---

---

## Notas

---

---

---

---

---

---

---

---

---

---

---

---