NORMAS DE PROGRAMACIÓN DE INFORMÁTICA INDUSTRIAL

1. Introducción

En este documento se presenta un estándar de escritura de programas en C y C++. El objetivo es facilitar el desarrollo de software en grupos (dirigidos por un director de proyecto o programador principal), fomentar la reutilización de código de proyectos anteriores, facilitar la documentación final y promover el intercambio de librerías. Para ello se establecen una serie de reglas que favorecen la claridad del código.

En este documento se utiliza el término **proyecto**, para referirse a un producto software, bien sea un programa o una librería. Por otra parte, se recomienda que los desarrollos se orienten a la construcción de librerías de propósito específico, para facilitar la distribución de la codificación y fomentar la reutilización de software.

Este estándar es independiente de la plataforma de desarrollo, es decir, no depende de la máquina, sistema operativo o compilador utilizado. Se desea que las reglas propuestas se apliquen de forma obligatoria, aunque en ocasiones, algunas normas pueden ser de utilización opcional (pero recomendada).

2. Estructura de programas

Salvo que un programa sea excesivamente pequeño, en general, se intentará dotar al código de una estructura que permita fácilmente su compilación o adaptación en otra plataforma.

Se tenderá a aislar el código de la función principal en un solo fichero y organizar el resto de funciones/clases en otros ficheros. La organización en distintos ficheros responderá a una agrupación funcional. Para cada agrupación funcional se establecerá un fichero de cabecera que puede servir de enlace para uno o varios ficheros de código en caso de que un solo fichero de código sea excesivamente grande y que exista la posibilidad de diferenciar operaciones según alguna característica.

En C++ se utilizará un fichero de cabecera y un fichero de código específico para cada una de las clases definidas.

Así, si por ejemplo el programa que se está realizando utiliza operaciones con números complejos y con matrices, una estructura lógica para un programa en C sería la siguiente.

- Principal.c : Fichero que contiene el main
- Matrices.c : Fichero con operaciones básicas de matrices (suma, producto, determinante)
- MatricesAvanzadas.c : Fichero con operaciones avanzaas de matrices (factorizacion LU, LR,…)
- Matrices.h : Fichero de cabecera para todas las operaciones de matrices públicas
- Complejos.c : Fichero que implementa las funciones para operar con números complejos
- Complejos.h : Fichero de cabecera para las funciones públicas de operación con complejos.

En el caso de C++ una estructura posible sería la siguiente:

- Principal.c : Fichero que contiene el main
- Matrices.cpp : Fichero con la implementación de la clase matriz.
- Matrices.h : Fichero de definición de la clase matriz
- Complejos.cpp : Fichero con la implementación de la clase complejo.
- Complejos.h : Fichero de definición de la clase complejo.

Esta simplificación es debida a que en C++, mediante la herencia, se tienen mecanismos sencillos que permiten ampliar la funcionalidad de una clase, por lo que las operaciones avanzadas (si fuera necesario) se podrían realizar creando una nueva clase derivada de la clase matriz.

Es conveniente separar en ficheros distintos las funciones que sirven de interfaz con el usuario (impresión o petición de datos) de aquellas que operan con datos pero que no necesitan interaccionar con el usuario.

3. Variables Globales

En general no se utilizarán variables globales.
La excepción más común a esta regla es el uso de una variable global que sirva para comunicar errores de ejecución.
En C++ existen mecanismos para lograr que una variable sea visible desde varias clases pero siempre bajo un cierto control mediante el modificador static. En cualquier caso su uso queda restringido a la transmisión de estados o al ahorro de memoria, nunca debe servir como medio de comunicación entre funciones o instancias.

4. Estructura interna de los archivos en C

Los ficheros que componen el programa seguirán una estructura predefinida que permitirá a cualquier usuario seguir fácilmente el código. Cada una de las secciones será separada por una línea en blanco como mínimo:

Estructura de un archivo de código fuente en C

1. Cajetín archivo fuente (Se explica en el apartado de comentarios).
2. Inclusión de los archivos de encabezamiento de las bibliotecas standard ('#include')
3. Archivos de encabezamiento de otras bibliotecas usadas.
4. Archivos de encabezamiento del propio del proyecto.
5. Constantes definidos exclusivamente para este archivo en orden alfabético (#define).
6. Macros definidas exclusivamente para este archivo en orden alfabético.
7. Prototipos de las funciones o procedimientos de las funciones no públicas de este archivo en orden alfabético.
8. Definición de las funciones. Cada función o procedimiento estará precedida de un cajetín con información sobre ella (Se explica en el apartado de comentarios).

Ejemplo:

```
/*****************************************************************
CABECERA DEL FICHERO archivo.c
*****************************************************************/

#include <stdio.h>
#include <math.h>
#include <X11.h>

#include "../tarjetaIO.h"
#include <sound.h>

#include "matriz.h"

#define PI 3.1415
#define N_E  2.178281

#define CUADRADO(X) ((X)*(X))

int esDiagonal(matriz *a);
int esSingular(matriz *b);

/*************************************************************
Cajetín de la primera funcion
*************************************************************/
void multiplicaMatriz(matriz mA,matriz mB, matriz mRes )
{
int i, j k;
.....
.....
}
```


```
/*************************************************************
Cajetín de la siguiente función
*************************************************************/
int EsDiagonal(matriz *mA)
{
…..
}

/*** y así con el resto de funciones***/
```

Estructura de un archivo de código fuente en C++

Salvo algunas excepciones, la estructura del código es análoga a la anterior

1. Cajetín archivo fuente (Se explica en el apartado de comentarios).
2. Inclusión de los archivos de encabezamiento de las bibliotecas standard ('#include')
3. Archivos de encabezamiento de otras bibliotecas usadas.
4. Archivos de encabezamiento del propio del proyecto.
5. Constantes definidas exclusivamente para este archivo en orden alfabético (#define).
6. Macros definidas exclusivamente para este archivo en orden alfabético.
7. Inicialización de las variables estáticas de la clase.
8. Implementación de los métodos inline no definidos en el fichero de cabecera.
9. Implementación de los métodos de la clase. Cada método estará precedido de un cajetín con información sobre ella (Se explica en el apartado de comentarios).
10. Implementación de las funciones amigas de la clase.

Estructura de un archivo de cabecera de C:

1. Cajetín del archivo de encabezamiento.
2. Directivas de exclusividad (explicado a continuación)
3. Archivos de encabezamiento standard.
4. Archivos de encabezamiento de otras bibliotecas.
5. Archivos de encabezamiento de este proyecto.
6. Definición de constantes .
7. Definición de estructuras.
8. Definición de Macros.
9. Variables externas (si no se han incluido en otro archivo de encabezamiento)
10. Prototipos de los procedimientos y funciones públicos (en orden alfabético), si no se han incluido en otro archivo de encabezamiento.
11. #endif necesario para cerrar el punto 2.

Estructura de un archivo de cabecera de C++:

1. Cajetín del archivo de encabezamiento.
2. Directivas de exclusividad (explicado a continuación)
3. Archivos de encabezamiento standard.
4. Archivos de encabezamiento de otras bibliotecas.
5. Archivos de encabezamiento de este proyecto.
6. Definición de constantes .
7. Definición de estructuras.
8. Definición de Macros.
9. Variables externas (si no se han incluido en otro archivo de encabezamiento)
10. Definición de la clase.
11. #endif necesario para cerrar el punto 2.

Para evitar errores de compilación por redefinición de identificadores, el contenido de los archivos de encabezamiento debe estar contenido en un bloque de directivas condicionales de de exclusividad de la forma:

```
#ifndef __NombreArchivoH__
#define   __NombreArchivoH__

...
 Aquí se continúa con el fichero de cabecera
...

#endif /*para __NombreArchivoH */
```

7. Identificadores

Tipos definidos por el usuario: clases, estructuras, enumeraciones.

Los identificadores de tipos definidos por el usuario se escribirán como sigue:

- Comenzarán con letra mayúscula.
- Si están formados por varias palabras, la primera letra de cada palabra se pondrá en mayúsculas.
- No se utilizarán números ni '_'.
- Se tenderá a utilizar sustantivos y adjetivos.
- Ejemplos: MatrizAvanzada, Complejo, DialogoCalculadora.

Variables o atributos.

Los identificadores de variables se escribirán como sigue:

- Comenzarán con letra minúscula.
- Si están formados por varias palabras, la primera letra de cada palabra se pondrá en mayúsculas excepto la primera.
- Los identificadores de una letra, normalmente i,j,k,l, están reservados para contadores. El resto de identificadores se recomienda que sean más extensos.
- No se utilizarán números ni '_'.
- Se tenderá a utilizar sustantivos y adjetivos.
- Ejemplos: estadoPrograma, minimo, maximo, colorFrontal, valorMax.

Macros y Constantes.

Las macros y constantes se escribirán como sigue:

- Se escribirán enteramente en mayúsculas
- Si están formadas por varias palabras, estas se separarán por medio de '_'
- Las macros tendrán los argumentos y el resultado con paréntesis en la evaluación de la expresión.
- Ejemplo: #define CUADRADO(x) ((x)*(x))

  #define DIMENSION_MAX 10

Funciones y métodos

Las funciones y métodos se escribirán como sigue:

- Comenzarán con letra minúscula.

- Si están formados por varias palabras, la primera letra de cada palabra se pondrá en mayúsculas.
- No se utilizarán números ni '_'.
- Se tenderá a utilizar la estructura verbo-sustantivo.
- Las funciones de interfaz de atributos de una clase utilizarán la notación inglesa get y set.
- Se utilizarán identificadores de los argumentos en los prototipos que faciliten la identificación del parámetro.
- Ejemplos: getColor, imprimirMatriz, calcularSolucion, etc.

7 .Comentarios o información interna del código

Los comentarios no deben salirse de un ancho de 80 columnas, que es normal en la mayor parte de las pantallas.

Debe quedar claro de un vistazo qué es código, y qué es comentario.
Los comentarios al código que sean extensos, se situarán antes de la línea que se comenta y alineados a su altura.

Existen dos elementos pertenecientes a los comentarios obligatorios en la documentación interna del código. Estos son los cajetines de información de las funciones y/o métodos y de los ficheros. Los cajetines deben comenzar con el comentario y seguir con una línea completa de asteriscos. Los distintos campos se titularán en mayúsculas. El fin del cajetín se realizará con una línea de asteriscos y el fin de comentario.

El **cajetín del fichero** deberá contener como mínimo la siguiente información:

- Nombre del fichero
- Autores
- Fecha de creación
- Descripción.

Opcionalmente se pueden –y se deben- añadir más: Ultimas modificaciones y responsable de dichas modificaciones, Funciones incluidas, librerías o objetos externos que utiliza.

Ejemplo de cajetín sencillo:

```
/*****************************************************************************
FICHERO: matriz.c
AUTOR:   Sebastián Lopez
FECHA:   17.II.2004
DESCRIPCION: Implementación de las funciones de la librería de matrices.
             El fichero de enlace es matriz.h.
*****************************************************************************/
```

El **cajetín de cada función o método** contendrá:

- Prototipo del procedimiento.
- Argumentos por orden, indicando el tipo y descripción.
- Valores de retorno, su tipo y significado.
- Descripción.
- Anotaciones: limitaciones de la función, suposiciones de las que parte (p.ej, que la memoria ha sido asignada, o que la debe liberar el que llama a la función).

Ejemplo de cajetín de función:

```
/*********************************************************************
FUNCION: int invertirMatriz(Matriz in, Matriz out)
ARGUMENTOS:
        Matriz in: Matriz que se quiere invertir.
        Matriz out:Matriz donde se sobrescribe la inversa si existe.

RETORNO: Retornará un 1 en caso de que exista la matriz inversa,
        y un  cero si la matriz es singular.
DESCRIPCION: Calcula la inversa de la matriz in y la guarda en out si
        tiene éxito. Informa de la existencia de la inversa.
*********************************************************************/
```

8. Otras normas acerca de la sintaxis en C/C++.

1. Después de cada signo de puntuación, "," o ";" se debe dejar un espacio, pero nunca antes.

2. Al escribir un comentario debe dejarse un espacio entre el * de apertura y el texto, y otro entre el texto y el * de cierre

```
/* Esto es un comentario. */
// Esto es un comentario
```

3. No debe dejarse un espacio después de abrir un paréntesis ni antes de cerrarlo.

4. Delimitación de bloques. Las llaves "{" que abren el cuerpo de un bucle, ya sea if, while, for o do-while, se situarán debajo de la sentencia y con un sangrado adicional de forma que queden alineadas con el código que depende de estas sentencias. La llave que cierra el bloque se situará en la línea siguiente a la última línea del bloque y con un nivel de sangrado igual al de las sentencias que encierra.

```
if(valor==5)
  {
  printf("Correcto");
  valor=valor*10;
  }
```

5. Al concatenar un if con el código else de un if anterior se hará de la siguiente manera:

```
if ( valor>3)
  {
  valor*=2;
  if(valor>255)valor=255;
  }
else  if ( valor>0 )
  {
  valor=3;
  printf("Ahora valor vale 3");
  }
else
  {
  valor=0;
  printf("Valor corregido a cero");
  }
```

6. Forma de switch case:

```
    switch(caracter)
      {
      case '0':
        printf("Cero");
```


```
      break;
      case '1':
      case '2':
      case '3':
        printf("Mayor que cero");
        printf("Menor que cuatro");
      break;
      default:
        printf("Fuera de rango");
      };
```

En caso de que el switch contenga muchos casos excluyentes y con una instrucción por caso, se puede adoptar una sintaxis más reducida como la siguiente:

```
switch(caracter)
  {
  case '0': valor=0; break;
  case '1': valor=1; break;
  case '2': valor=2; break;
  case '3': valor=3; break;
  case '4': valor=4; break;
  case '5': valor=5; break;
  case '6': valor=6; break;
  case '7': valor=7; break;
  default: valor=-1;
  }
```

7. Debe haber una línea blanca entre la definición de variables locales y el cuerpo de un procedimiento.

8. El código del procedimiento debe comenzar a escribirse al mismo nivel de sangrado que la llave de apertura, que se escribirá al mismo nivel de sangrado que el nombre del procedimiento.

9. Se recomienda usar secciones en el código de un procedimiento, para separar partes de un procedimiento que realicen labores distintas. La primera línea de cada sección debería ser un comentario explicando el por qué de esa sección.