

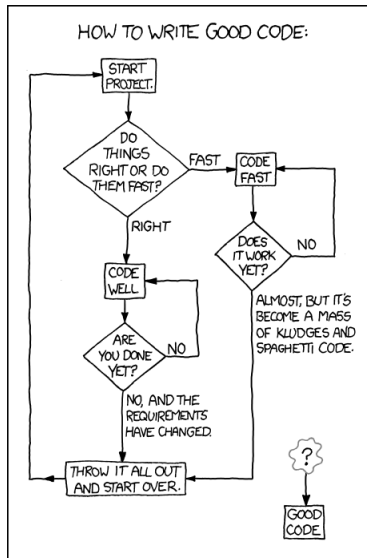
Buenas prácticas de programación en C

Leopoldo Agorio

Curso PIE 2021

27 de abril de 2021

- 1 Introducción
- 2 Organización y ensayo del código
- 3 Legibilidad y estilo
- 4 Específicos de C
- 5 Comentando un código
- 6 Doxygen



- En esta clase iremos más allá de únicamente hacer que un código funcione.
- “Buenas prácticas”: *conjunto de reglas informales que la comunidad ha aprendido para mejorar la calidad del software.*
- El objetivo es facilitar el **proceso de desarrollo**, y el de **posterior mantenimiento** o mejora, así como la **lectura tanto por autores como terceros**.

- 1 Introducción
- 2 Organización y ensayo del código
 - Diseño modular
 - Ensayo módulo a módulo
 - El uso del printf
- 3 Legibilidad y estilo
- 4 Específicos de C
- 5 Comentando un código
- 6 Doxygen

- Pensar cuidadosamente cómo estructurar el código antes de empezar a escribir.

```
1 int invertir(Matriz33 A, Matriz33 Ainv)
2 {
3     Matriz33 Mcof, Mcoft;
4
5
6     float detA = determinante(A);
7
8     if (detA == 0)
9         return (-1);
10
11     matrizCofactores(A, Mcof);
12
13
14
15
16
17
18     transponer(Mcof, Mcoft);
19
20
21
22
23     matrizPorValor(Mcoft, 1/detA, Ainv);
24
25     return 0;
26 }
```

```
1 int invertir(Matriz33 A, Matriz33 Ainv)
2 {
3     Matriz33 Mcof, Mcoft;
4     int i, j;
5
6     float detA = determinante(A);
7
8     if (detA == 0)
9         return (-1);
10
11     /*armar matriz de cofactores*/
12     for(i=0; i<3; i++){
13         for(j=0; j<3; j++){
14             Mcof[i][j] = (A[(i+1)%3][(j+1)%3]*A[(i+2)%3][(j+2)%3])
15                 - (A[(i+1)%3][(j+2)%3]*A[(i+2)%3][(j+1)%3]);
16         }
17     }
18     /*trasponer*/
19     for (i = 0; i < 3; i++)
20         for (j = 0; j < 3; j++)
21             Mcoft[i][j] = Mcof[j][i];
22
23     /*dividir entre determinante*/
24     for(i=0; i<3; i++)
25         for(j=0; j<3; j++){
26             Ainv[i][j] = Mcoft[i][j] / detA;
27         }
28     return 0;
29 }
```

- Uno suele tender a escribir el código completo y luego empezar a debuggear y chequear errores.
- Esto sirve para proyectos chicos, pero para proyectos grandes no.
- **Debuggear cada módulo que completan** evita tener que perder tiempo en descubrir qué es lo que está mal.
- Lo mejor es que algunos **casos de prueba se planifiquen incluso antes de empezar** a tirar código, y se lleven adelante mientras se desarrolla el código.

- Suele ser útil para debuggear **imprimir avisos o variables en momentos clave** del proceso.
- Es bueno que este **texto sea descriptivo** y diga **en que lugar del código** me encuentro. “llegué al segundo while de invertir, i vale %d ” es mejor mensaje que “flag” o “%d” .
- Esto es útil para **detectar dónde ocurren los problemas**, que es parte principal del proceso de debuggeo.
- De todas formas lo más correcto es usar el debugger (gdb/nemiver) que lo verán más adelante.

El uso del printf - DEBPRINT

- Una técnica usual más elegante que ir comentando y descomentando prints es el DEBPRINT, que permite dejar en su programa los prints del proceso de desarrollo y hacer que no impriman nada con un define:

```
1 #include <stdio.h>
2 #include "imagen.h"
3
4 #define DEBUG
5 #ifdef DEBUG
6     #define DEBPRINT(...) fprintf(stderr, __VA_ARGS__)
7 #else
8     #define DEBPRINT(...)
9 #endif
10
11 CodigoError inicializarImagen(Imagen* pnew, int columnas, int filas,
12                               int valorMaximo, TipoImagen tipo){
13     if(.....){
14         ....
15
16         .....
17
18         .....
19
20         DEBPRINT("inicializarImagen: se retornó sin errores\n");
21         return OK;
22     }
23     else{
24         DEBPRINT("inicializarImagen: ERROR en parámetros de entrada.\n");
25         return ERROR;
26     }
27 }
```

- 1 Introducción
- 2 Organización y ensayo del código
- 3 Legibilidad y estilo
 - Uso de nombres descriptivos
 - Indentación
 - Largo de líneas
 - Consistencia interna
- 4 Específicos de C
- 5 Comentando un código
- 6 Doxygen

- Para variables, funciones y constantes usar nombres **descriptivos, tratando de evitar nombres demasiado largos.**
- “*matrizCofactores*” es un buen nombre. “*funcaux*” no, “*rutinaParaCalcularMatrizDeCofactoresAntesDelInvertir*” tampoco.
- Está bueno mantener **convenciones usuales** como usar *i,j* para contadores.
- A lo que hay que aspirar es a que su código se pueda leer en voz alta **casi como si leyeran una oración.**

Uso de nombres descriptivos

```
1 int invertir(Matriz33 A, Matriz33 Ainv)
2 {
3     Matriz33 Mcof, Mcoft;
4     float detA = determinante(A);
5     if (detA == 0)
6         return (-1);
7
8     matrizCofactores(A, Mcof);
9     transponer(Mcof, Mcoft);
10    matrizPorValor(Mcoft, 1/detA, Ainv);
11
12    return 0;
13 }
--
```

- Cuerpos de funciones, loops, declaraciones if/else, etc, deben ir indentados. Declaraciones al mismo nivel deben indentarse la misma cantidad.

```
1 int algoritmo(int x, int y) {
2   declaracion1;
3   declaracion2;
4   while (criterio) {
5     for(i=0;i<N;i++)
6       recorrerarreglo;
7     llamarfuncion;
8     incrementarvariable;
9     if(situacionborde) {
10      opcion1;
11      continuacion;
12    } else {
13      opcion2;
14    }
15  }
16  procesodecierre;
17 }
```

- No deben haber líneas de más de 80 caracteres. Su editor debe tener una forma de ayudarlos a cumplir este criterio.

```
1 int a = /*en C las sentencias las termina el ; y no los enter, por lo que */
2 b = c = d /*pueden partir una línea larga poniendo enter. Para partir un */
3 = f; /*comentario como este en varias líneas mantengan columna de inicio*/
4
5 if ( ((vec[i] < 0 ) && (vecprima[j] > 234))
6      || ((vec[j] == 3456) && (vecprima[i] <= 4444))
7      || ((vec[i] > 10) && (vecprima[j] == 3333))
8      )
9 {
10     sentencia1;
11     sentencia2;
12 }
13
14
15 printf("El criterio es que ninguna línea llegue a 80 caracteres. Pueden");
16 printf("configurar su editor para que les senale la marca de 80 como acá.\n");
17
18 printf("Esta línea y las anteriores sirven de ejemplo de como partir"
19        "un print en varias líneas de código");
20
```

- Es bueno **mantener un único criterio** para definiciones de función, nombres de variables, comentarios, etc.
- *derivar_respecto_y* es un nombre de función tan válido como *derivarRespectoY*, a menos que ya tuvieran una función de nombre *derivar_respecto_x*.

- 1 Introducción
- 2 Organización y ensayo del código
- 3 Legibilidad y estilo
- 4 Específicos de C
 - Manejo de errores
 - Alocación de memoria
 - Constantes y variables globales
 - Propósito de un main
- 5 Comentando un código
- 6 Doxygen

- **Usar detección y manejo de errores.**
- **Chequear el retorno** de funciones que puedan indicar error y **manejarlos apropiadamente.**

- Si reservan memoria dinámica (malloc), recuerden liberarlo luego.
- **1 malloc 1 free.**

Constantes y variables globales

- Las **variables globales se deben evitar**. La forma correcta es buen uso de pase por referencia para cambiar valor de variables.
- Las constantes hacen el código más legible y más fácil de modificar.
- **SPOD - único punto de definicion**. Si quiero cambiar alguna variable general debería tener que cambiarla una única vez, y no andar buscando todas las veces que la usé.

```
1 /*-----*/           /*-----*/
2 /*   Hagan esto:   */   /*   Eviten esto:   */
3 /*-----*/           /*-----*/
4 #define MAX    50
5
6 int  buf[MAX];
7 if( i < MAX) { ... }

int  buf[50];
if( i < 50) { ... }
```

- **En un producto terminado**, el main tiene que servir como paneo general y de alto nivel de su solución, sin entrar en detalles de bajo nivel.
- Esto no quita que realicen rutinas de ensayo/debuggeo de sus diferentes subpartes del código.

- 1 Introducción
- 2 Organización y ensayo del código
- 3 Legibilidad y estilo
- 4 Específicos de C
- 5 Comentando un código
 - Para cada archivo
 - Para cada función
 - Para líneas difíciles
- 6 Doxygen

- En el apuro, el comentado del código es la primer cosa que uno tiende a relegar.
- Para cualquier trabajo en equipo, el comentado del código es **esencial** para explicar el código.
- Comentar bien un código también permite re-utilizarlo y re-acondicionarlo en un futuro. Leer un código sin comentarios 1 año después es muy difícil.

- Cada .h y .c debe tener un comentario de alto nivel describiendo el contenido, e indicando fecha y nombre del autor.

```
*matriz.c x
1
2/**
3 * Archivo: matriz.c
4 * Autor: Leopoldo Agorio
5 * Fecha: 11 feb 2021
6 * Resumen: Código fuente obligatorio 1.
7 *
8 * Código fuente implementando las funciones del obligatorio 1. Se trata de una
9 * biblioteca de matemática sencilla en matrices de 3x3 y vectores de 3x1.
10 */
11
12#include <stdio.h>
13#include "matriz.h"
```

- Cada función (tanto en .h como .c) debe tener un comentario que describa que hace, que parametros toma, que variables retorna.
- En .h el comentario es para el usuario de la interfaz, en .c es para desarrolladores. Se estila que el .c sea más descriptivo de los detalles del algoritmo.
- Se puede escribir el comentario de función *antes* que el código para guiar el trabajo.

Comentario de función

```
15 /**
16  * Resumen: Llena todos los elementos de un vector con un mismo valor.
17  *
18  * Ejemplo de uso - llenar vec con el número 3.14 - :
19  *
20  * Vector3 vec;
21  * llenarVector(3.14, vec);
22  *
23  *
24  * parametro value: Valor con el que llenar el vector.
25  * parametro V: Vector a llenar.
26  * return: Nulo.
27  */
28 void llenarVector(float value, Vector3 V){
29     int i;
30     for(i = 0; i < 3; i++)
31         V[i] = value;
32 }
```

Comentario de línea

- Secuencias complicadas, entreveradas o feas deben tener comentarios en línea aclarando qué hacen. (si no se pudo evitar con buen criterio de modularización y nombres)
- Es importante comentar las partes complicadas, pero no excederse. Comentar cosas obvias (“acá a i le sumo 1”) puede esconder las verdaderamente importantes.

```
158 int invertir(Matriz33 A, Matriz33 Ainv){
159     int i,j;
160     float det = determinante(A);
161     if(!det){
162         printf("ERROR AL INVERTIR, DETERMINANTE NULO\n");
163         return -1;
164     }else{
165         for(i=0;i<3;i++)
166             for(j=0;j<3;j++) /*se usa formula de cofactores traspuesta*/
167                 Ainv[j][i] = ((A[(i+1)%3][(j+1)%3] * A[(i+2)%3][(j+2)%3])/det
168                     - ((A[(i+1)%3][(j+2)%3]*A[(i+2)%3][(j+1)%3])/det);
169         return 0;
170     }
171 }
172
```

- 1 Introducción
- 2 Organización y ensayo del código
- 3 Legibilidad y estilo
- 4 Específicos de C
- 5 Comentando un código
- 6 Doxygen

BONUSTRACK

Doxygen ($D0_2$) – Qué es?



The screenshot shows the Doxygen website homepage. The browser address bar displays "doxygen.nl/index.html". The website has a blue header with the "Doxygen" logo and navigation tabs for "Casa", "Descargas", "Documentación", "Extensiones", and "Apoyo". A sidebar on the left lists various links under the "Doxygen" heading, including "Acerca de", "Descargas", "Registro de cambios", "Documentación", "Involucrarse", "Lista de deseos", "Ejemplos de", "Enlaces", "Extensiones", "Apoyo", "Informar errores", and "Donar". The main content area features the heading "Doxygen" and a sub-heading "Genera documentación a partir del código fuente". Below this, a paragraph explains that Doxygen is a standard tool for generating documentation from C++ annotated sources, and is also compatible with other languages like C, Objective-C, C#, PHP, Java, Python, IDL, Fortran, and VHDL. A list of three ways Doxygen can be used is provided: 1. Generating in-line HTML documentation or out-of-line RTF/PostScript/PDF manuals from source code. 2. Configuring Doxygen to extract source file structures for large projects, including dependency graphs and inheritance diagrams. 3. Using Doxygen to create standard documentation like user manuals and website content. At the bottom, it notes that Doxygen is developed for Mac OS X and Linux but is portable to Windows.

Doxygen

Genera documentación a partir del código fuente

Doxygen es la herramienta estándar de facto para generar documentación a partir de fuentes C++ anotadas, pero también es compatible con otros lenguajes de programación populares como C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft y UNO / OpenOffice), Fortran, VHDL y hasta cierto punto D.

Doxygen puede ayudarlo de tres maneras:

1. Puede generar un navegador de documentación en línea (en HTML) y / o un manual de referencia fuera de línea (en RTF) a partir de un conjunto de archivos fuente documentados. También hay soporte para generar resultados en RTF (MS-Word), PostScript, PDF con hipervínculos, HTML comprimido y páginas de manual de Unix. La documentación se extrae directamente de las fuentes, lo que hace que sea mucho más fácil mantener la documentación coherente con el código fuente.
2. Puede configurar doxygen para extraer la estructura del código de archivos fuente no documentados. Esto es muy útil para encontrar rápidamente su camino en grandes distribuciones de fuentes. Doxygen también puede visualizar las relaciones entre los distintos elementos mediante la inclusión de gráficos de dependencia, diagramas de herencia y diagramas de colaboración, que se generan todos automáticamente.
3. También puede usar doxygen para crear documentación normal (como hice para el manual de usuario y el sitio web de doxygen).

Doxygen está desarrollado bajo Mac OS X y Linux, pero está configurado para ser altamente portátil. Como resultado, también se ejecuta en la mayoría de los otros tipos de Unix. Además, hay disponibles ejecutables para Windows.

BONUSTRACK

Doxygen ($D0_2$) – Qué se obtiene?

The image shows two side-by-side browser windows displaying Doxygen-generated documentation for a project named 'PIE Oblig 1'.

The left window shows the main documentation page for 'PIE Oblig 1'. It includes a navigation bar, a search bar, and a section titled 'Referencia del Archivo matriz.h'. Below this, there is a diagram showing the class hierarchy: 'matriz.h' is the parent, with 'matriz.c' and 'obligacion.c' as children. The page also lists 'typedefs' and 'Funciones' (functions) with their signatures and brief descriptions.

The right window shows the 'Índice general' (General Index) page, which is a table of contents for the entire documentation. It lists sections and their corresponding page numbers:

- 1 Índice de archivos 1
 - 1.1 Lista de archivos 1
- 2 Documentación de archivos 3
 - 2.1 Referencia del Archivo matriz.c 3
 - 2.1.1 Descripción de la función 4
 - 2.1.2 Documentación de las funciones 4
 - 2.1.2.1 determinante() 4
 - 2.1.2.2 imprimirMatriz() 5
 - 2.1.2.3 invertir() 5
 - 2.1.2.4 leerMatriz() 6
 - 2.1.2.5 leerVector() 6
 - 2.1.2.6 mostrarVector() 7
 - 2.1.2.7 mostrarMatriz() 7
 - 2.1.2.8 mostrarVector() 8
 - 2.1.2.9 transponer() 8
 - 2.2 Referencia del Archivo matriz.h 9
 - 2.2.1 Descripción de la función 10
 - 2.2.2 Documentación de las funciones 10
 - 2.2.2.1 determinante() 10
 - 2.2.2.2 imprimirMatriz() 11
 - 2.2.2.3 imprimirVector() 11
 - 2.2.2.4 invertir() 12
 - 2.2.2.5 leerMatriz() 13
 - 2.2.2.6 leerVector() 14
 - 2.2.2.7 mostrarVector() 14
 - 2.2.2.8 mostrarVector() 15
 - 2.2.2.9 transponer() 16
- Índice 19

BONUSTRACK

Doxygen ($D0_2$) Cómo comentar su código para doxygen?

Descripción detallada

Código fuente obligatorio 1.

Autor

Leopoldo Agorio

Fecha

11 feb 2021 Código fuente implementando las funciones del obligatorio 1. Se trata de una biblioteca de matemática sencilla en matrices de 3×3 y vectores de 3×1 .

Definición en el archivo `matriz.c`.

Documentación de las funciones

`\det`

```
3
4 /**
5  * @file matriz.c
6  * @author Leopoldo Agorio
7  * @date 11 feb 2021
8  * @brief Código fuente obligatorio 1.
9  *
10 * Código fuente implementando las funciones del obligatorio 1. Se
11 * trata de una
12 * biblioteca de matemática sencilla en matrices de  $3 \times 3$  y vectores de
13 *  $3 \times 1$ .
14 */
15 #include <stdio.h>
16 #include "matriz.h"
```

BONUSTRACK

Doxygen ($D0_2$) Cómo comentar su código para doxygen?

invertir()

```
int invertir ( Matriz33 A,  
            Matriz33 Ainv  
            )
```

Inversión de una matriz de 3x3.

Se usa la fórmula explícita de inversión en 3x3 es decir, en cada elemento determinante de autoadjunta traspuesta sobre determinante de la matriz entera.

Ejemplo de uso:

```
Matriz33 inv;  
trasponer(mat, inv);
```

Parámetros

A Matriz a Invertir.
Ainv Matriz donde guardar resultado.

Devuelve

Nulo.

Definición en la línea 166 del archivo matriz.c.

```
149  
150 /**  
151  * @brief Inversión de una matriz de 3x3.  
152  *  
153  * Se usa la fórmula explícita de inversión en 3x3 es decir, en cada  
154  * elemento  
155  * determinante de autoadjunta traspuesta sobre determinante de la  
156  * matriz entera.  
157  *  
158  * Ejemplo de uso:  
159  * @code  
160  * Matriz33 inv;  
161  * trasponer(mat, inv);  
162  * @endcode  
163  * @param A Matriz a invertir.  
164  * @param Ainv Matriz donde guardar resultado.  
165  * @return Nulo.  
166  */  
167 int invertir(Matriz33 A, Matriz33 Ainv){
```

BONUSTRACK

Doxygen ($D0_2$) Cómo correrlo?

```
Open [icon] comandosdoxygen.txt Save [icon] [icon] [icon]
~/Documents/OnlyLeopoldo/Docencia/PIE/dropbox/2021/Documentacion
1 instalar doxygen (si no lo tienen ya):
2 $ sudo apt install doxygen
3
4 armar doxygen:
5 $ doxygen -g do2config
6
7 abrir el archivo do2config y editar a gusto. Para tener título, que quede en
8 español y que permita ver código fuente de archivos .c (esto no necesariamente
9 es deseable), por ejemplo poner:
10
11 -PROJECT_NAME = "PIE Oblig 1" (línea 35)
12 -OUTPUT_LANGUAGE = Spanish (línea 94)
13 -INPUT = matriz.h matriz.c obligatorio1.c (línea 793/867)
14 -SOURCE_BROWSER = YES (línea 998/1077)
15
16 correr doxygen:
17 $ doxygen do2config
18
19 Usar salida HTML:
20 - Entrar a la carpeta "html" generada y abrir con mozilla "index.html"
21
22 Usar salida Latex:
23 - Entrar a la carpeta "latex"
24 $ make pdf
25 - Abrir refman.pdf
```


- **Best coding practices**
<https://slideplayer.com/slide/13134831/>
- **C Code Style Guidelines:**
https://www.cs.swarthmore.edu/~newhall/unixhelp/c_codestyle.html
- **Doxygen reference site**
<https://www.doxygen.nl/index.html>
- **Doxygen usage example (for C)**
http://fnch.users.sourceforge.net/doxygen_c.html

¿ Preguntas ?