Buenas prácticas de programación en C

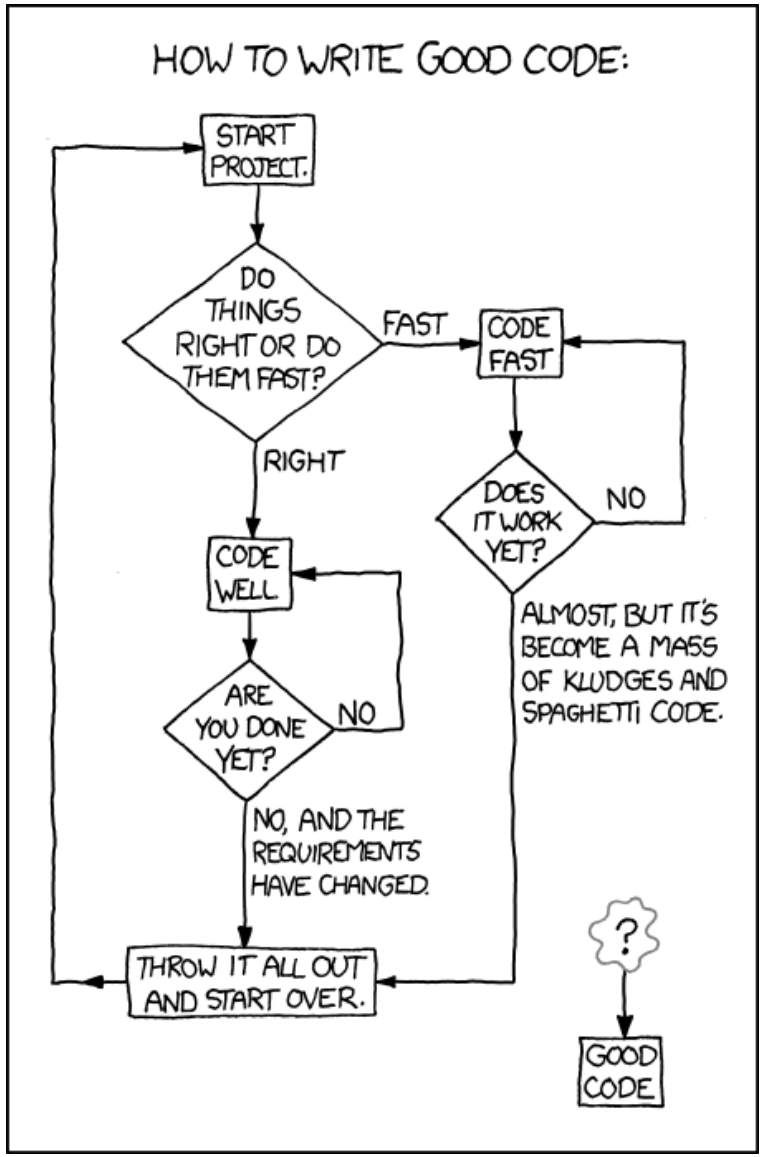
Leopoldo Agorio

Curso PIE 2021

27 de abril de 2021

Resumen

Introducción



Introducción

- 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**.

Resumen

Diseño modular

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

```c
int invertir(Matriz33 A, Matriz33 Ainv)
{
    Matriz33 Mcof, Mcoft;


    float detA = determinante(A);

    if (detA == 0)
        return (-1);

    matrizCofactores(A, Mcof);









    transponer(Mcof, Mcoft);




    matrizPorValor(Mcoft, 1/detA, Ainv);

    return 0;
}
```

```c
int invertir(Matriz33 A, Matriz33 Ainv)
{
    Matriz33 Mcof, Mcoft;
    int i,j;

    float detA = determinante(A);

    if (detA == 0)
        return (-1);

    /*armar matriz de cofactores*/
    for(i=0; i<3; i++)
        for(j=0; j<3; j++){
            Mcof[i][j] = (A[(i+1)%3][(j+1)%3]*A[(i+2)%3][(j+2)%3])
                        -(A[(i+1)%3][(j+2)%3]*A[(i+2)%3][(j+1)%3]);
        }

    /*trasponer*/
    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
            Mcoft[i][j] = Mcof[j][i];

    /*dividir entre determinante*/
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            Ainv[i][j] = Mcoft[i][j] / detA;

    return 0;
}
```

Ensayo módulo a módulo

- 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.

El uso del printf

- 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 }
```

Resumen

Uso de nombres descriptivos

- Para variables, funciones y constantes usar nombres **descriptivos, tratando de evitar nombres demasiado largos**.
- "*matrizCofactores*" es un buen nombre. "*funcaux*" no, "*rutinaParaCalcularMatrizDeCofactoresAntesDeInvertir*" 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 }
```

Indentación

- 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 }
```

Largo de líneas

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

```c
int a =   /*en C las sentencias las termina el ; y no los enter, por lo que */
b = c = d /*pueden partir una línea larga poniendo enter. Para partir un */
= f;      /*comentario como este en varias lineas mantengan columna de inicio*/

if ( ((vec[i] < 0 ) && (vecprima[j] > 234))
     || ((vec[j] == 3456) && (vecprima[i] <= 4444))
     || ((vec[i] > 10) && (vecprima[j] == 3333))
   )
{
    sentencia1;
    sentencia2;
}


printf("El criterio es que ninguna línea llegue a 80 caracteres. Pueden");
printf("configurar su editor para que les senale la marca de 80 como acá.\n");

printf("Esta línea y las anteriores sirven de ejemplo de como partir"
       "un print en varias líneas de código");

```

Consistencia interna

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

Resumen

Manejo de errores

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

Alocación de memoria

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

```
/*------------------*/
/*   Eviten esto:   */
/*------------------*/


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

Propósito de un main

- **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.

Resumen

Idea general

- 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.

Comentario de archivo

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

*matriz.c

```
 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"
```

Comentario de función

- 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
```

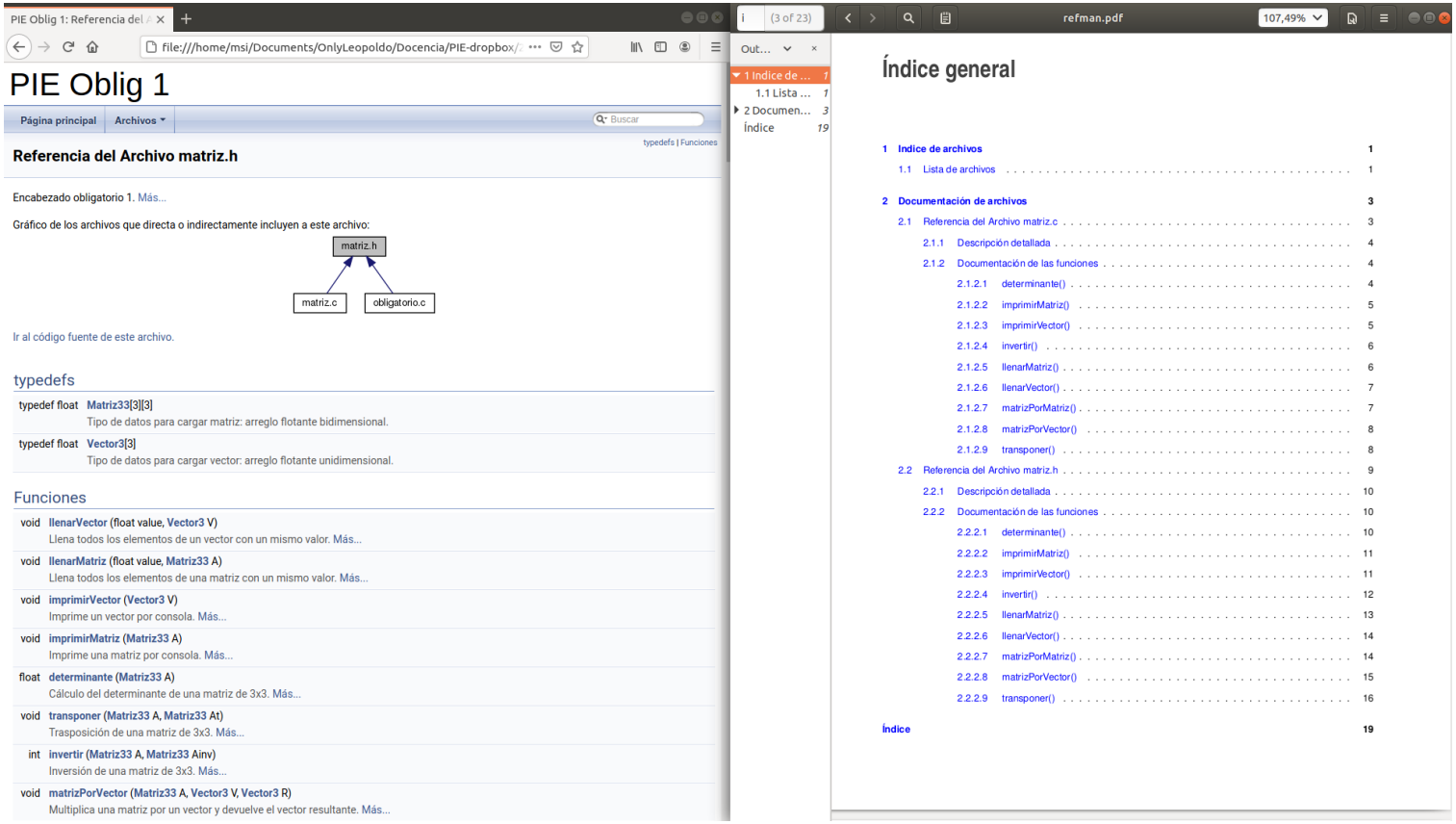
Resumen

BONUSTRACK

Doxygen ($D0_2$) – Qué es?

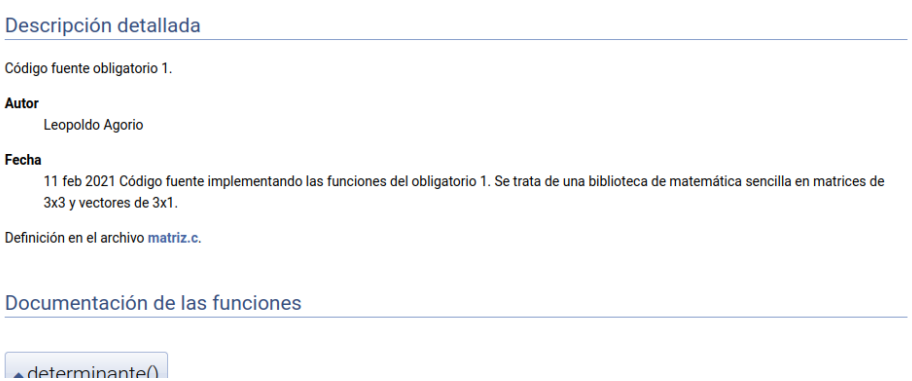
BONUSTRACK

Doxygen ($D0_2$) – Qué se obtiene?



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 3x3 y vectores de 3x1.

Definición en el archivo matriz.c.

Documentación de las funciones

determinante()

```
 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
    trata de una
11  * biblioteca de matemática sencilla en matrices de 3x3 y vectores de
    3x1.
12  */
13
14 #include <stdio.h>
15 #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
     elemento
154  * determinante de autoadjunta traspuesta sobre determinante de la
     matriz entera.
155  *
156  * Ejemplo de uso:
157  * @code
158  * Matriz33 inv;
159  * trasponer(mat, inv);
160  * @endcode
161  *
162  * @param A Matriz a invertir.
163  * @param Ainv Matriz donde guardar resultado.
164  * @return Nulo.
165  */
166 int invertir(Matriz33 A, Matriz33 Ainv){
```

BONUSTRACK

Doxygen ($D0_2$) Cómo correrlo?

Open ▾ | comandosdoxygen.txt — ~/Documents/OnlyLeopoldo/Docencia/PIE-dropbox/2021/Documentacion | Save

```
instalar doxygen (si no lo tienen ya):
$ sudo apt install doxygen

armar doxygen:
$ doxygen -g do2config

abrir el archivo do2config y editar a gusto. Para tener título, que quede en
español y que permita ver código fuente de archivos .c (esto no necesariamente
es deseable), por ejemplo poner:

-PROJECT_NAME = "PIE Oblig 1" (linea 35)
-OUTPUT_LANGUAGE        = Spanish (linea 94)
-INPUT                  = matriz.h matriz.c obligatorio1.c (linea 793/867)
-SOURCE_BROWSER         = YES (linea 998/1077)

correr doxygen:
$ doxygen do2config

Usar salida HTML:
- Entrar a la carpeta "html" generada y abrir con mozilla "index.html"

Usar salida Latex:
- Entrar a la carpeta "latex"
$ make pdf
- Abrir refman.pdf
```

Referencias:

- **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 ?