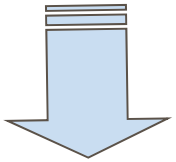
Documentación

Eva Lucrecia Gibaja Galindo

Dpto. Informática y Análisis Numérico

Introducción

- Un programa es escrito por un programador, pero será consultado, a lo largo de la vida del mismo, por otras personas o programadores
- Un programa puede contener errores que pueden pasar desapercibidos. Es posible que en un futuro sea revisado y modificado por otro programador
- A lo largo de la vida del programa, el cliente o usuario puede proponer una serie de modificaciones, o nuevas opciones. El código habrá de ser revisado (por el programador que lo implementó o por otra persona)



Mantenimiento

Documentación

- Con el tiempo se olvidan y pierden muchos detalles relacionados con un programa
- La documentación consiste en describir *lo que hace* el programa y *como lo hace*
  - **Documentación interna**. La documentación aportada en el propio código
  - **Documentación externa**. Aportada fuera del código del programa (manuales de usuario, etc.)

Documentación interna. Comentarios

- Todo lenguaje de programación proporciona la forma de incluir líneas de comentarios en el mismo (/**/, //, !, <!-- - ->)
- Los editores utilizan generalmente un color distinto para las líneas de comentario
- Si un algoritmo en pseudo-código incorpora comentarios, estos deben ser trasladados al código del programa
- Tampoco es conveniente que sean excesivos (Ej. Un comentario en cada línea)

Documentación interna. Comentarios

- **Cabecera de las funciones o procedimientos**
  - Nombre, Propósito, Autor
  - Parámetros de entrada y salida
  - Procedimientos o funciones a los que llama y Procedimientos o funciones por las que es llamado
  - Códigos de error
  - Fecha de creación, Fecha de modificaciones

```
/*************************************************************/
   Nombre: buscarporNombre.
   Tipo: int.
   Objetivo: Visualiza todos los registros que tengan un nombre dado
             por el usuario.

   Parametros de entrada:
      - char *fichero: Nombre del fichero.
      - char *auxNombre: Nombre de los registros a visualizar.
   Precondiciones: El fichero ha de existir.
   Valor devuelto: 0 si no se encuentra ningún registro y 1 si se
                   encuentra alguno.
   Funciones a las que llama:
      - escribirDatosPersonales.
   Fecha de creación: 7-01-03.
   Autor:
/*************************************************************/
int buscarporNombre(char *fichero, char *auxNombre)
{
 //Cuerpo de la función
}
```

Documentación interna. Comentarios

- **Declaraciones de variables o tipos compuestos**
  - Asignar un nombre adecuado a su función (autodocumentación). Si no es posible, incluir comentario aclaratorio de su misión
- **Esquemas condicionales e iterativos**
  - Clarificar su propósito, condiciones de salida o permanencia, etc
- **Llamadas a subprogramas o funciones**
  - Indicar la función y efecto que tienen sobre las variables usadas por la misma

```
struct DatosPersonales
{
  char nombre[15]; //nombre
  char apellido[15]; //apellido
  long dni; //dni
};
```

Documentación interna. Presentación

- El **sangrado** o indentación de párrafos
  - Utilizar distintos niveles de sangrado, desplazando a la derecha los bloques de código que estén incluidos en otros (esquemas iterativos y condicionales anidados.
  - Facilita la comprensión de código
  - Si existen muchos niveles de anidamiento, se deben reducir los sangrado.
- Hacer corresponder cada línea de código con una línea física del programa
- Espacios en blanco
- **Líneas en blanco** para separar los bloques (bucles, condicionales, procedimientos, etc.) de un programa
- **Agrupamiento** de sentencias de E/S
- Correspondencia entre el orden de ubicación de los bloques y el orden en que van a ser ejecutados

Ejemplos de documentación interna

- Nombre de las variables

```
get a b c
if a < 24 and b < 60 and c < 60
return true
else
return false
```

```
get horas minutos segundos
if horas < 24 and minutos < 60
and segundos < 60
return true
else
return false
```

Ejemplos de documentación interna

- Indentación

```
if(horas<24&&minutos<60&&segundos<60)
{return
true;}
else{return false;}
```

```
if(horas < 24 && minutos < 60 && segundos < 60)
{
   return true;
}
else
{
   return false;
}
```

Ejemplos de documentación interna

- Espaciado

```
int cuenta;
for(cuenta=0;cuenta<10;cuenta++){printf("%d"
,cuenta*cuenta+cuenta);}
```

```
int cuenta;
for (cuenta = 0; cuenta < 10; cuenta++)
{
   printf("%d", cuenta * cuenta + cuenta);
}
```

Documentación externa

- **Manual de usuario**
  - Documento destinado a facilitar el uso del programa a los usuarios del mismo
  - Debe reflejar una explicación detallada, con gráficos, de todas las opciones de la aplicación
  - Debe contener un ejemplo práctico de uso del programa que utilice todas las opciones del mismo
- **Manual del operador**
  - Destinado al operador informático. Contiene las especificaciones que se han de cumplir, para el correcto funcionamiento del programa (instalación, entorno de funcionamiento, requerimientos hardware)
- **Manual de mantenimiento**
  - Destinado al programador o programadores que se dediquen al mantenimiento del programa
- **Especificaciones del programa**
  - Elaborado por el programador, refleja las especificaciones encargadas por el cliente, y que ha de cumplir el programa

Documentación externa

- **Lista de datos de prueba y resultados**
  - Refleja algunas de las pruebas realizadas al programa. Intentar cubrir todos los casos posibles de funcionamiento
- **Historia de desarrollo del programa y modificaciones posteriores**
  - Realizado por el que implemente la aplicación y por los encargados del mantenimiento
- **Diseño descendente con detalle de módulos y submódulos**
  - Siguiendo alguna metodología específica
- **Versiones del programa**
  - Reflejando las diferencias, ventajas e inconvenientes existentes entre las posibles versiones de la aplicación

Los principios del programador

- **Principio del Carácter Personal**
  - Escribe tu código de forma que refleje, y saque a relucir, solo lo mejor de tu carácter personal
- **Principio de la Estética**
  - Esfuérzate por conseguir la belleza y la elegancia en cada aspecto de tu trabajo
- **Principio de la Claridad**
  - Dale el mismo valor a la claridad que a la corrección. Utiliza activamente técnicas que mejoran la claridad del código. La corrección vendrá casi por sí sola

Los principios del programador

- **Principio de la Distribución**
  - Usa la distribución visual de tu código para comunicar la estructura de tu código a un lector humano
- **Principio de lo Explicito**
  - Intenta siempre favorecer lo explícito sobre lo implícito
- **Principio de Código Auto-Documentado**
  - La documentación más fiable para el software es el propio código. En muchos casos, el propio código es la única documentación. Por lo tanto, esfuérzate en hacer que tu código sea auto-documentado, y allí donde no sea posible, añade comentarios

Los principios del programador

- **Principio de los Comentarios**
  - Comenta mediante frases completas para resumir y comunicar la intención
- **Principio de las Suposiciones**
  - Da los pasos que sean razonables para comprobar, documentar y prestar atención a las suposiciones hechas en cada módulo y rutina
- **Principio de la Interfaz con el Usuario**
  - Nunca hagas que el usuario se sienta estúpido

Los principios del programador

- **Principio de Volver Atrás**
  - El momento de escribir buen código es justamente el preciso momento en el que lo estás escribiendo
- **El Principio de El Tiempo y El Dinero de Otros**
  - Un verdadero profesional no gasta el tiempo ni el dinero de otras personas produciendo software que no esté razonablemente libre de fallos; que no esté mínimamente probado; que no cumpla con los requisitos establecidos; que esté falsamente adornado con características innecesarias; o que tenga un aspecto deplorable

Notas finales

- **Dijkstra** (uno de los autores más reconocido en ciencias de la computación) recomienda diez páginas de comentarios por cada página de código!!. Ello da una idea de la importancia que tiene la documentación
- Se trata de **recomendaciones**, no hay por qué llevarlas al extremo, depende de la complejidad del programa


LAS AVENTURAS DEL CAPITAN LITERAL
HOY PRESENTAMOS: "PROGRAMADOR"
...LA FUNCIÓN A TIENE EL OBJETO... SE LO PASA POR REFERENCIA A LA FUNCIÓN B... ÉSTA RECIBE... SE ABRE POR LA BANDA Y CARACOLEA ANTE LA PRESENCIA DE UN IF....
...SI SEÑOOOR... QUE MAESTRÍA, ÉSTO MERECE UN PURITO PEICH... ATENCIÓN QUE LA FUNCIÓN C ESTA DESMARCADA,... RECIBE...VA DIRECTA CON EL OBJETO HACIA LA CAPA DE PERSISTENCIA...
....SORTEA A UN BUCLE FOR... NO, A DOS... NADIE LE MARCA... ENCARA EN SOLITARIO HACIA LA BASE DE DATOS... VA A INSERTAR...VA A INSERTAR... VA A INSERTAAAR Y...¡NO HOMBRE NO!
EL COLEGIADO SEÑALA SQLEXCEPTION... LAS FUNCIONES SE LE ECHAN ENCIMA PROTESTANDO LA DECISIÓN Y RAZÓN NO LES FALTA...
MALDITA LA HORA EN QUE LE DIJIMOS QUE EN ESTA EMPRESA ERA MUY IMPORTANTE COMENTAR BIEN EL CÓDIGO
...PORQUE LA REPETICION EN EL DEBUGGER MUESTRA CLARAMENTE QUE NO ERA... NO ERA EXCEPCIÓN, ALMA DE CÁNTARO... LA TABLA EXISTIA Y ESTABA EN POSICIÓN REGLAMENTARIA....
FANTASMITAS.COM

Herramientas para documentar código

- **Java**
  - Javadoc
- **C**
  - Doxygen
- **.NET**
  - Ndoc
- **Documentar a partir de un diseño**
  - Rational Rose
- **Para casi cualquier tipo de lenguaje**
  - Doc-O-Matic

¿Qué es Doxygen?

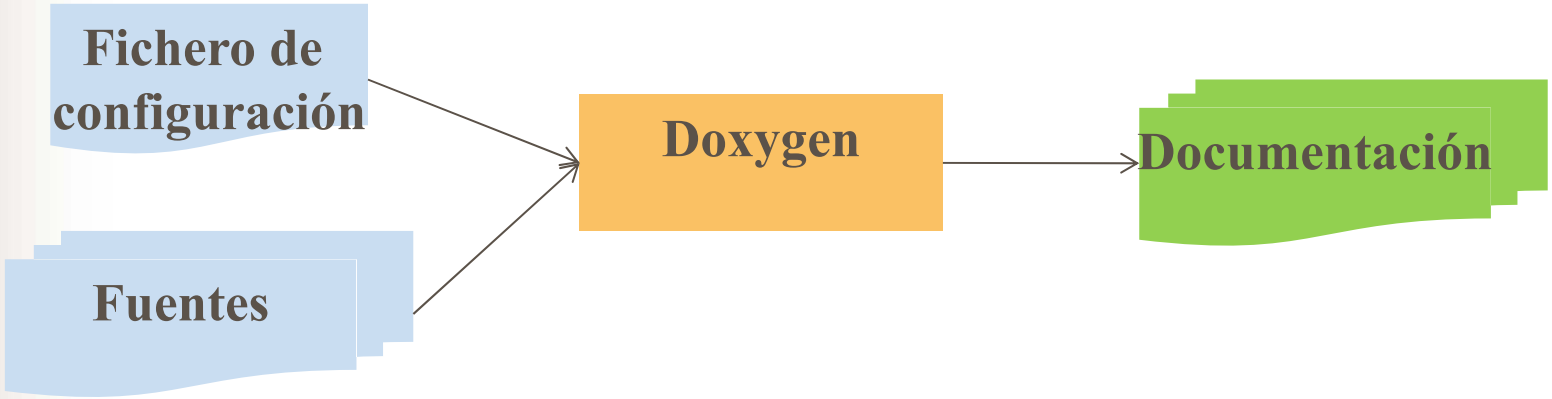
- Doxygen es un programa generador de documentación de códigos fuente
- Soporta C, C++, Java, Python, IDL, Fortran, VHDL, PHP, etc.
- Bajo licencia GNU
- Formatos de salida:
  - Documentación HTML
  - LaTeX
  - Pdf
  - Páginas de manual

Instalación

- Bajar archivos de www.doxygen.org
- Linux, Windows y otros
- Una vez instalado se debe configurar el PATH, para que encuentre los ejecutables
- Su operación es mediante consola o interfaz gráfica

Utilizar Doxygen

1. Generar un fichero de configuración
   - **`doxygen -g ficheroConfiguracion`**
2. Editar el fichero de configuración con nuestras preferencias.
3. Generar la documentación a partir del fichero de configuración.
   - **`doxygen ficheroConfiguracion`**



Fichero de configuración

- PROJECT_NAME = *Nombre del Proyecto*
- PROJECT_NUMBER = *Versión*
- OUTPUT_DIRECTORY = *Directorio del resultado*
- OUTPUT_LANGUAGE = *Spanish (o English)*
- INPUT = *Directorio con los códigos o lista de ficheros separados por espacios*
- FILE_PATTERNS = *Tipos de archivos si INPUT contiene directorios*
- SOURCE_BROWSER = *Muestra el código de los ficheros .c*
- GENERATE_HTML = *YES*
- GENERATE_LATEX = *NO*
- GENERATE_RTF = *NO*
- GENERATE_MAN = *NO*
- GENERATE_XML = *NO*

Comentarios en doxygen

- **Comentarios en C**
  - Línea simple: `// Comentario de una línea`
  - Varias líneas

```
/* … línea 1 …
… línea 2 … */
```

- **Comentarios en doxygen**
  - La documentación del código se realiza incluyendo en el código bloques de comentarios
  - Doxygen permite varios tipos de bloques de comentarios:
    - Estilo *C* (con extra *) o *Qt* (con extra !)

Comentarios Doxygen	Estilo C	Estilo Qt

Varias líneas	`/**`
	`... texto ...`
	`*/`
		`/*!`
		`... texto ...`
		`*/`

Una línea	`///`
	`/// ... texto ...`
	`///`
		`//!`
		`//! ... texto ...`
		`//!`

Comentarios en el código

- Los comentarios puede ir antes o después del bloque de código al que se refiere

	Antes	Después
Una línea	`///Funcion que suma dos enteros` `int suma (int a, int b);`	`int suma (int a, int b);` `///<Funcion que suma dos enteros`
Varias líneas	`/**Funcion que suma dos enteros*/` `int suma (int a, int b);`	`int suma (int a, int b);` `/**<Funcion que suma dos enteros*/`

¿Qué se comenta?

- ¿Qué hay que comentar en estilo doxygen?
- Información sobre el fichero – programa
  - Las funciones
  - Las estructuras definidas
  - Los defines
  - Las variables globales
- ¿Qué información incluir en la documentación?
  - Descripción breve y/o detallada
  - Autor, fecha
  - Información específica sobre lo que estemos documentando
    - Parámetros y valor devuelto para las funciones
    - Campos para las estructuras, etc.

Descripciones breve y detallada

- Solo es válido incluir una descripción breve (*brief*) y una detallada:
  - Tipo 1 (sin dejar una línea en blanco)

```
///<Descripcion breve
/**<Descripcion detallada*/
```

  - Tipo 2 (utilizando el comando *brief*)

```
/** @brief Descripcion breve
La desc. breve continua hasta encontrar linea en blanco

La desc. detallada empieza al encontrar linea en blanco
*/
```

- Descripción de los comandos Doxygen:
  http://www.stack.nl/~dimitri/doxygen/commands.html

Comandos. Estructurales

- Los comandos pueden estar precedidos de \ o @
  - En prácticas los ejemplos están montados con \
- Comandos estructurales
  - **@struct** <para documentar una estructura>
  - **@fn** <para documentar una función>
  - **@var** <para documentar una variable>
  - **@def** <para documentar un define>
  - **@typedef** <documenta un typedef>

Comandos. Estructurales

```
/**
@struct punto
@brief Definicion breve de una estructura de tipo punto
*/
struct punto
{
float x; /**<Coordenada x del punto (D. larga)*/
float y; ///<Coordenada y del punto (D. breve)
};
```

```
/**
@typedef t_punto
@brief Definicion breve del typedef

Definicion detallada del typedef
*/
typedef struct punto t_punto;
```

```
/**
@def PI
@brief Definicion breve del numero PI

Definicion detallada del nuero PI
*/
#define PI 3.14
```

Comentarios. Ficheros

- Dentro de los comentarios de doxygen se incluyen comentarios
- Cabecera del fichero:
  - **@author** <comentario sobre el autor>
  - **@date** <fecha>
  - **@file** <comentario sobre el fichero>
  - **@version** <version del archivo>

Comentarios. Ficheros

```
/**
 @file  myfile.h
 @brief descripción breve sobre el fichero
 @author Eva Gibaja
 @date 13-04-2012
 @version 1.0

Este fichero contiene el código de las funciones
necesarias para trabajar con imagenes:
@li Cargar imagen
@li Grabar imagen
@li Espejo horizontal
*/
```

Comandos. Funciones

- Comandos para documentar funciones
  - **@fn** <nombre o prototipo de la funcioon>
  - **@param** <comentario de un parámetro>
  - **@return** <comentario sobre el valor devuelto>
  - **@post** <postcondicion de la funcion>
  - **@pre** <precondición de la función>

```
/**
 @brief Descripción breve de la funcion
 @fn Nombre o prototipo de la función
 @param a  Descripcion del parámetro a
 @return  Información sobre el valor devuelto
*/
```

Comandos. Funciones

```
/* Nombre:restaCuadrados
Tipo: entero (int)
Objetivo: calcula la diferencia de los cuadrados
de dos números (n1^2-n2^2)
Parámetros entrada:
int n1: primer numero
int n2: segundo numero
Precondiciones: Ninguna
Valor devuelto: n1^2-n2^2
Utiliza: cuadrado
Autor: Maria Luque
Fecha: 20-11-2008
*/
int restaCuadrados(int n1, int n2);
```

Comandos. Funciones

```
/**
@fn restaCuadrados(int n1, int n2)
@brief Calcula la diferencia de dos cuadrados
@param n1 Primer numero
@param n2 Segundo numero
@pre Ninguna
@return @f$ n_1^2-n_2^2$
@author Maria Luque
@date 20-11-2008

Esta funcion calcula la diferencia entre los
cuadrados de dos números pasados como argumentos
*/
int restaCuadrados(int n1, int n2);
```

Comandos. Modificar aspecto

- Comandos que modifican el aspecto de la documentación

Comando	Función
`@n`	Salto de línea
`@b`	Pone en negrita la siguiente palabra
`@em`	Pone en cursiva la siguiente palabra
`@p`	Pone en estilo curier la siguiente palabra
`\@, \\, \$, \&, \#, \%, \<, \>`	Escribe los símbolos @, \, $, &, #, %, <, >
`@li <descripción de un item>`	Genera un lista de argumentos. Cada elemento de la lista empieza con @li
`@f$`	Incluir una fórmula de texto (formato LaTex)
`@f[`	Comienzo de una fórmula larga (formato LaTex)
`@f]`	Fin de una fórmula larga (formato LaTex)

Ejemplo

```
/**
  @file 3.1.c
  @brief Este archivo es un ejemplo
  @author Eva Gibaja
  @date  26/08/2011
 */
#include "3.2.h“
/**
  @fn void main ()
  @brief funcion main()
  @return nada
*/
void main () {
  int i;

  boolean c;
  c = (char) i;
  suma((int)c,5);
  return;
}
/**
 @mainpage Introduccion

En esta pagina podemos incluir una descripcion general
*/
```

Ejemplo

```
/**
   @file 3.2.c
   @brief Este archivo es otro ejemplo
*/

#include "3.2.h"

int suma (int a, int b) {
   return a+b;
}
```

Ejemplo

```
/**
  @file 3.2.h
  @brief Este archivo es otro ejemplo

  Esta es la descripcion detallada del archivo
*/
#define Pi 3.14        /**<Definicion detallada de la constante pi */
typedef int booleano; /**<Documentacion detallada de un typedef*/
/**
 @struct punto
 @brief Definicion de una estructura de tipo punto
*/
struct punto
{
   float x; /**<Coordenada x del punto descripcion detallada*/
   float y;  ///<Coordenada y del punto descripcion breve
};

/**
 @fn void suma (int a, int b)
 @brief Funcion que suma
 @param a sumando 1
 @param b sumando 2
 @return Devuelve la suma a+b
*/
int suma(int a, int b);
```

Recursos sobre Doxygen

- Página de doxygen:
  - Comentarios en doxygen: http://www.stack.nl/~dimitri/doxygen/docblocks.html
  - Descripción de los comandos doxygen: http://www.stack.nl/~dimitri/doxygen/commands.html
  - Incluir fórmulas: http://www.stack.nl/~dimitri/doxygen/formulas.html
  - Manual: http://www.stack.nl/~dimitri/doxygen/manual.html
- Otros recursos:
  - Manual muy sencillo: http://trevinca.ei.uvigo.es/~jgarcia/FP/manuales/manualDoxygen.pdf
  - Tutorial: http://class.ece.iastate.edu/cpre288/lecture/Doxygen/DoxygenTutorial.pdf
  - Otro tutorial: http://www.scenebeta.com/tutorial/documentando-el-codigo-con-doxygen