

8 - Documentazione del software

Documentare un prodotto

Un prodotto viene documentato tramite due manuali solitamente:

- **Manuale Tecnico**: descrive **come è fatto** un prodotto
- **Manuale d'uso**: descrive **cosa fa** un prodotto

Un prodotto quindi non è altro che un insieme di attributi tangibili e intangibili di un bene o un servizio volti a procurare un beneficio a un utilizzatore, ottenuto tipicamente attraverso un processo di produzione

Alcune delle parole evidenziate possono essere associate al software, che anche lui è un **prodotto**:

- **Attributi** = Funzionalità del software
- **Servizio** = Scopo del software
- **Utilizzatore** = Utente del software
- **Produzione** = Sviluppo del software

Ogni prodotto che si rispetti ha un **interfaccia** e un **implementazione**:

- L'interfaccia è come il prodotto viene mostrato al pubblico
- L'implementazione invece è la struttura interna

La documentazione dell'interfaccia spiega come può il software (o la libreria) essere **utilizzata da un utente finale**, mentre la documentazione sull'implementazione fornisce dettagli **sul processo alla realizzazione del prodotto**

Documentazione di una libreria

Anche una **libreria** è un modulo di software più grande e può essere vista come un prodotto:

- **Attributi** = Cosa implementa (funzioni, procedure, tipi di dati, costanti...)
- **Servizio** = Scopo della libreria
- **Utilizzatore** = Sviluppatore che usa la libreria
- **Produzione** = Programmazione della libreria

L'interfaccia viene rappresentata dalle **funzioni implementate in una libreria**, mentre l'implementazione è **il codice sorgente che le realizza**, e tutte e due possono essere documentate in modo generativo.

L'interfaccia di una libreria software rappresenta **la descrizione degli attributi pubblici**, cioè le funzionalità utilizzabili da un utente, in questo caso per le librerie in C si definisce con un file header di un modulo.

Esempio:

```
#ifndef MATH_UTILS_H
#define MATH_UTILS_H

// Funzione per calcolare il quadrato di un numero intero
int square(int x);

// Funzione per calcolare il massimo tra due numeri interi
int max(int a, int b);

#endif // MATH_UTILS_H
```

Fornire comunque il codice dell'interfaccia non è **sufficiente a rendere utilizzabile una libreria**, ma è necessaria una **documentazione aggiuntiva**.

Cosa deve offrire questa documentazione aggiuntiva?:

- Descrivere il **significato dei tipi** e delle **costanti pubbliche**
- Descrivere **cosa fanno le funzioni e le procedure**
- **Non deve descrivere come le funzioni e le procedure svolgono il loro compito** (concetto di information hiding)

Documentare il codice equivale a scrivere un **manuale d'uso**, in questo caso l'interfaccia di una libreria

Documentazione vs commento

C'è un'importante differenza tra la documentazione di un prodotto sorgente e il commento del codice sorgente:

- La **documentazione** spiega il significato di una funzione **all'utilizzatore**
- Il **commento** spiega i punti più critici al **programmatore** della funzione

Documentazione di una funzione

Quando si documenta una funzione cosa bisogna inserire?

1. Cosa fa la funzione:

- Scenari tipici (comportamento normale della funzione)
- Scenari eccezionali (cosa fa la funzione in caso di situazioni anomale)

2. Significato e dominio dei parametri

3. Esempi d'uso

4. Pre-condizioni

5. Post-condizioni

6. Eventuali riferimenti (altre funzioni, link, test, standard...)

7. Eventuali note (funzioni deprecate, portabilità...)

Documentazione pre-condizione

La pre-condizione è **una condizione che deve essere vera prima della chiamata di una funzione**, solitamente è una condizione sui parametri.

Se una pre-condizione è falsa, **il funzionamento non è garantito**.

ES. la funzione per il calcolo del BMI ha come pre-condizione che peso e altezza siano maggiori di 0 ed eventualmente minori di una certa soglia.

Il chiamante di una procedura ha la responsabilità di garantire la verifica di tutte le condizioni (una pre-condizione falsa è sintomo di bug)

IMPORTANTE: la definizione delle pre-condizioni non implica il mancato uso della programmazione difensiva, infatti è bene specificare le precondizioni ma anche rendere il codice solido in modo tale da restituire risultati anche quando le pre-condizioni non sono verificate

Sapere a priori quali errori possono verificarsi è diverso dallo scrivere codice in grado di gestirli!

Documentazione post-condizione

La post-condizione è **una condizione che deve essere vera dopo l'esecuzione di una funzione**, solitamente è sui parametri di output.

La falsificazione di una post-condizione (se le pre-condizioni sono verificate) è un chiaro sintomo della presenza di un bug nella funzione.

ES. riprendendo il BMI, se non corrisponde a quello corretto evidentemente c'è un problema nell'implementazione

Documentazione in linea

Oltre a documentare le funzioni nel modo convenzionale, si può documentare una libreria nel **codice sorgente stesso**, ma è necessario che tale documentazione sia poi accessibile all'utilizzatore della libreria

La documentazione in linea è scritta mediante **commenti speciali** (doc-comments) che seguono una convenzione particolare

Generazione automatica di documentazione

Esistono tool particolari che esaminano il codice sorgente alla ricerca di doc-comment, poi vengono formattati in documenti in formato pdf, html, etc.

Questi strumenti si chiamano **tool di generazione automatica della documentazione**

Noi andremo ad usare [Doxygen](#)

Doxygen

Formattazione commenti

I commenti in Doxygen hanno una struttura leggermente differente, serve a farli distinguere dai commenti base di C

- Commenti “semplici” in C

```
// singola linea
```



Commenti **Doxygen** in C

```
/// singola linea
```

```
/*
 * multi
 * linea
 */
```



```
/**
 * multi
 * linea
 */
```

Header file

Una caratteristica di Doxygen è l'utilizzo di particolari annotazioni (tag) che servono a indicare porzioni particolari del codice sorgente, `@file` ad esempio indica il nome del file, `@version` la versione. L'uso dei tag serve a dare un significato ai commenti

```
* @file documentazione.h

* Synthetic description.
*
* Detailed description here.
* @version 0.1
* @date 17/mag/2018
* @authors c. musto

* @copyright GNU Public License.
* @bug Not all memory is freed when deleting an object.
* @warning Improper use can crash your application
*/
```

#define e tipi di dato

Nei commenti inseriamo informazioni sul significato della `#define` e sulle caratteristiche del tipo di dato

```

/**                                     /**
 * Max admissible value.              * 32-bit unsigned integer.
 */                                   */
#define MAX_VALUE 100                typedef unsigned int UINT32;

```

Enumerazione

```

/** An enum type.
 * Detailed documentation
 */
enum EnumType
{
    int EVa11,      /**< enum value 1 */
    int EVa12       /**< enum value 2 */
};

```

Variabili globali

Nel caso delle variabili globali, si usa il tag `@warning` per fornire eventuali informazioni legate all'utilizzo

```

/**
 * Contains the last error code.
 * @warning Not thread safe!
 */
int errno;

```

Funzioni

Utilizziamo `@param` per indicare quali parametri utilizza la funzione e `@return` per spiegare che valore restituisce. Prima di questi tag inseriamo anche una descrizione generale

```
/**  
 *   Opens a file descriptor.  
 *   This function opens a file on the file system .  
 *  
 *   @param[in] pathname The name of the descriptor  
 *   @param[in] flags Opening flags.  
 *   @return file handler.  
 */  
int open(const char* pathname, int flags);
```

Separare i commenti

A volte può essere utile distinguere i commenti, inserendo una parte di essi nel .h e una parte nel .c

Nel .h si inseriscono informazioni su parametri, tipi di ritorno e sullo scopo generale della funzione

Nel .c (dedicato all'implementazione) si dettagliano meglio le pre/post condizioni

```

/**
 * Calcola il codice di controllo di un numero ISBN.
 *
 * @param[in] isbn_num Il numero ISBN per il quale
calcolare il codice di
 *
 * controllo.
 * @param[in] isbn_length Il formato del codice ISBN (10
o 13).
 * @return il codice di controllo.
 */

char isbn_ctrlcode(char[] isbn_num, int isbn_length);

/**
 * La funzione calcola il codice di controllo secondo lo standard
ISO
 * (http://it.wikipedia.org/wiki/ISBN#Calcolo_.28ISBN-10.29).
 * La funzione restituisce il codice di controllo per codici ISBN
a 10 o 13 cifre.
 *
 * @pre isbn_num ha lunghezza pari a isbn_length - 1.
 * @pre isbn_num contiene solo cifre decimali
 * @pre isbn_length assume valore 10 o 13
 * @post il carattere restituito e' una cifra decimale oppure 'X'
 */
char isbn_ctrlcode(char[] isbn_num, int isbn_length) {
{
    int i = 0;

    char code [MAX] = {'\0'};

    ...

```

Manuale e configuration file

Il manuale è disponibile a questo [link](#)

Oggi progetto deve avere un Doxygen configuration file che contiene informazioni utili per la generazione della documentazione dell'intero progetto

Anche per la generazione della documentazione di un singolo file sorgente è necessario un Doxygen configuration file.

Un template di file sorgente può essere generato con il seguente comando: `doxygen -g <config-file>` (dove config-file è il nome del file di configurazione)

Il template andrà customizzato in base all'esigenze del progetto, possono essere utilizzati dei tool grafici per modificare facilmente il template: doxywizard