

TTK4235: Dokumentasjon med Doxygen

Kolbjørn Austreng

Vår 2020

Om øvingen

Dokumentasjon er en av de tingene man ikke setter ordentlig pris på før man jobber på et prosjekt som ikke er dokumentert godt nok. Når det skjer, så er ikke de grå hårene og den nihilistiske livsinnstillingen langt unna.

Derfor, for å unngå at dere alle blir sure knarker og kråker før tiden er inne, er det lurt å ikke bare “få ting til å funke”, men også dokumentere hvordan det fungerer i løpet av utviklingsprosessen.

Som mennesker har vi derimot én kritisk hindring som står i veien for vår egen suksess; dokumentasjon er ikke særlig gøy. Dermed, hvis kode og dokumentasjon er separate arbeidsoppgaver, vil dokumentasjonen gjerne glemmes litt bort.

Doxygen er en “minste motstands vei”-løsning på dette problemet. Tanken er at hvis du allikevel sitter og skriver kode, så kan du like gjerne putte på noen spesielt formaterte kommentarer rundt om kring, så kan Doxygen automatisk generere dokumentasjonen for deg. Da er det plutselig mye mer sannsynlig at dokumentasjonen

- a) Skrives i utgangspunktet, og
- b) Oppdateres hver gang koden endrer seg

Seksjon 1 til seksjon 1.4 forklarer bruken av Doxygen. Gå gjennom dette først. Deretter gjør dere oppgaven i seksjon 2 og viser frem resultatet på sal for godkjenning.

Som vanlig er fristen innen slutten av uke 7 (14. Februar).

1 Eksempelprosjekt

Dere har fått utlevert en mappe som heter “**example**”. Alt som har med intro til Doxygen å gjøre ligger der inne.

Vi har et prosjekt som består av en “memory”-modul, som definerer noen enkle operasjoner på lister av heltall. Denne modulen er implementert i `memory_library.h` og `memory_library.c`, og prosjekttreet vårt ser slik ut:

```
.
|-- [Makefile og eventuelle andre filer]
|
|-- source
|   |-- main.c
|   |-- memory_library.h
|   |-- memory_library.c
```

Det eneste modulen består av er funksjonene `memory_reverse_copy()` og `memory_multiply_elements()`. Det er disse vi ønsker å dokumentere.

1.1 Doxygen config

Når dere kjører Doxygen, forventer programmet å finne en konfigurasjonsfil som forteller hva det skal gjøre. For å opprette en konfigurasjonsfil kaller vi `doxygen -g doxconfig`. Dette vil opprette en fil kalt `doxconfig`, med en del parametre dere kan sette for å bestemme oppførselen til Doxygen senere.

Når vi har generert denne filen redigerer vi den med en teksteditor av fritt valg. Endringene vi gjør i dette eksempelet er som følger:

```
PROJECT_NAME = "Memory Library Example"
OPTIMIZE_OUTPUT_FOR_C = YES
INPUT = source/
SOURCE_BROWSER = YES
```

Alle parametre dere kan sette er forklart med kommentarer i konfigurasjonsfilen. Prosjekttreet vårt ser nå slik ut:

```
.
|-- [Makefile og eventuelle andre filer]
|-- doxconfig
|
|-- source
|   |-- main.c
|   |-- memory_library.h
|   |-- memory_library.c
```

1.2 Kommenter koden

For å dokumentere koden vår, legger vi inn spesielt formaterte kommentarer som består av kommandoer som sier hvordan kommentaren skal tolkes. I toppen av hver fil vi vil at Doxygen skal se gjennom, må vi inkludere `file`-kommandoen, for å inkludere den i dokumentasjonen. I utgangspunktet ser `memory_library.h` slik ut:

```
#ifndef MEMORY_LIBRARY_H
#define MEMORY_LIBRARY_H

int memory_reverse_copy(const int * p_from, int * p_to, int
↪ size);

void memory_multiply_elements(int * p_buffer, int factor, int
↪ size);

#endif
```

For å fortelle Doxygen at det skal genereres dokumentasjon for denne filen, må vi markere den med Doxygen sin `file`-kommando, som vi putter i toppen av `h`-filen:

```
/**
 * @file
 * @brief A simple library for doing operations on memory
 * buffers consisting of integers
 */
```

I tillegg til `@file`, har vi også spesifisert `@brief`, som simpelthen gir en kort oppsummering av denne filens funksjon.

Vi skal nå dokumentere hva de to funksjonene i filen gjør. Dokumentasjonen putter vi i `h`-filen, men her har dere implementasjonen av de to funksjonene for å skjønne hva de gjør:

```
#include "memory_library.h"
#include <stdlib.h>

int memory_reverse_copy(const int * p_from, int * p_to, int
↪ size){
    if(p_from == NULL || p_to == NULL){
        return 1;
    }

    for(int i = 0; i < size; i++){
```

```

        p_to[size - i - 1] = p_from[i];
    }
    return 0;
}

void memory_multiply_elements(int * p_buffer, int factor, int
↪ size){
    if(p_buffer == NULL){
        exit(1);
    }

    for(int i = 0; i < size; i++){
        p_buffer[i] *= factor;
    }
}

```

1.2.1 int memory_reverse_copy(...)

Som vi ser fra funksjonsdefinisjonen vil `memory_reverse_copy` ta inn to buffere, og kopiere fra det første inn i det andre, samtidig som rekkefølgen på elementene blir reversert. I pseudokode har vi altså:

```

array_one = {1, 2, 3, 4}
array_two = [space for four integers]
memory_reverse_copy(array_one, array_two, 4)
array_two == {4, 3, 2, 1}

```

For å dokumentere denne funksjonen, skriver vi følgende kommentar inn rett over `memory_reverse_copy` i h-filen:

```

/**
 * @brief Copy a list of integers from one buffer to another,
 * reversing the order of the output in the process.
 *
 * @param[in] p_from Source buffer.
 * @param[out] p_to Destination buffer.
 * @param[in] size Number of integers in the buffer.
 *
 * @return 0 on success, 1 if either @p p_from or @p p_to
 * is a @c NULL pointer.
 */

```

Som før gir `@brief` kun en kort oppsummering av hva funksjonen gjør. Kommandoen `@param` forteller hva en funksjonsparameter sin oppgave er. Det er frivillig å legge ved `[in]`, `[out]` eller `[in,out]` ved en parametererklæring; denne vil simpelthen fortelle om parameteren blir modifisert av funksjonen

eller ikke. Dette er nyttig for ting som kalles med referanse i C++, eller via pekere i C. Følgende konvensjon brukes:

- **[in]**: Parameteren brukes inne i funksjonen, men vil ikke endres på utsiden av funksjonskroppen.
- **[out]**: Parameteren brukes ikke direkte inne i funksjonen, men vil være endret på utsiden av funksjonen når den returnerer.
- **[in,out]**: Verdien til parameteren brukes direkte i funksjonen, og den vil være endret på utsiden av funksjonen når den returnerer.

Deretter ser vi kommandoen `@return`, som naturlig nok forteller oss hva funksjonen returnerer. Denne trenger man ikke spesifisere for **void**-funksjoner.

Vi ser også bruk av `@c`, og `@p`. `@c` er et hint til Doxygen om at det neste ordet er et kodeord. Altså vil i dette tilfellet “NULL” bli markert som kode i dokumentasjonen. `@p` er en referanse til en parameter, og vil også markeres som kode i den ferdige dokumentasjonen.

1.2.2 `void memory_multiply_elements(...)`

Det er ikke mye ny magi for å dokumentere denne funksjonen. Det eneste vi gjør forskjellig her er å introdusere `@warning`, som vi kan bruke for å advare brukeren mot oppførsel som kanskje ikke er åpenbar:

```
/**
 * @brief Multiply all the elements in @p p_buffer, of size
 * @p size with the supplied @p factor.
 *
 * @param[in,out] p_buffer Buffer of integers to be multiplied
 * with @p factor.
 *
 * @param[in] factor Factor to multiply each of the
 * elements in @p p_buffer with.
 *
 * @param[in] size Size of @p p_buffer.
 *
 * @warning If @p p_buffer is @c NULL, the function will
 * abruptly terminate the program with exit code @c 1.
 */
```

1.2.3 Heads up

For en komplett liste over hvilke kommandoer Doxygen støtter, kan dere ta en titt på doxygen.nl.

Legg også merke til at det er mulig å bruke “\” istedenfor “@” for å starte kommandoer i Doxygen. For C og C++ anbefales “@” fordi tegnet ikke kolliderer med *escape characters* (som \n eller \t) i koden, som gjør det enklere å søke gjennom etter Doxygen-kommentarer senere.



Det er lett å glemme at hver fil må markeres med `@file` i toppen for å bli inkludert. Da vil dere ende opp med tom dokumentasjon.

1.3 Generer dokumentasjonen

Fra samme mappe som konfigurasjonsfilen (“`doxconfig`”) ligger i, kaller vi `doxygen doxconfig`. Dette vil generere dokumentasjon i både HTML- og L^AT_EX-format, i hver sin mappe. Når dette er gjort, vil prosjektmappen se slik ut:

```
.
|-- [Makefile og eventuelle andre filer]
|-- doxconfig
|
|-- source
|   |-- main.c
|   |-- memory_library.h
|   |-- memory_library.c
|
|-- html
|   |-- index.html
|   |-- [...]
|
|-- latex
    |-- refman.tex
    |-- [...]
```

Dere kan velge hvilke formater Doxygen skal generere dokumentasjon på ved å endre på parametrene `GENERATE_LATEX` eller `GENERATE_HTML` i konfigurasjonsfilen.

Dere kan også endre hvor Doxygen plasserer den genererte dokumentasjonen, ved å endre på parameteren `OUTPUT_DIRECTORY`.

1.4 Nyt sexy dokumentasjon

Åpne `html/index.html` i en egnet nettleser ved å enten kalle noe i duren av `firefox html/index.html` fra terminalen, eller trykke `Ctrl + O` inne i nettleseren og åpne `index.html` derfra.

I utgangspunktet vil dere bli møtt av en ganske tom side, fordi vi ikke har brydd oss om å sette så mye under “**Project related configuration options**” i konfigurasjonsfilen. Dette står dere fritt til å endre på som dere vil. Uansett vil dere ha en lenke til **Files** oppe i venstre hjørne. Denne kan dere følge for å få en oversikt som illustrert i figur 1.




File List	
Here is a list of all documented files with brief descriptions:	
<div>▼ source</div>	
 <code>main.c</code>	The main file of the application
 <code>memory_library.c</code>	Implementation file for memory library
 <code>memory_library.h</code>	A simple library for doing operations on memory buffers consisting of integers

Figure 1: Oversikt over hvilke filer prosjektet består av.

Om dere herfra følger lenken til `memory_library.h`, vil dere få dokumentasjonen til denne filen; gjengitt i figur 2.

memory_library.h File Reference	
A simple library for doing operations on memory buffers consisting of integers. More...	
Go to the source code of this file.	
Functions	
int <code>memory_reverse_copy</code> (const int *p_from, int *p_to, int size)	Copy a list of integers from one buffer to another, reversing the order of the output in the process. More...
void <code>memory_multiply_elements</code> (int *p_buffer, int factor, int size)	Multiply all the elements in p_buffer, of size size with the supplied factor. More...

Figure 2: Dokumentasjon for `memory_library.h`.

Herfra kan dere også følge lenkene som heter “**More...**” for å få dokumentasjonen til hver enkelt funksjon. Dette er gjengitt i figur 3 og figur 4.

◆ memory_reverse_copy()

```
int memory_reverse_copy ( const int * p_from,  
                          int *      p_to,  
                          int      size  
                          )
```

Copy a list of integers from one buffer to another, reversing the order of the output in the process.

Parameters

[in] **p_from** Source buffer.
[out] **p_to** Destination buffer.
[in] **size** Number of integers in the buffer.

Returns

0 on success, 1 if either p_from or p_to is a NULL pointer.

Definition at line 8 of file [memory_library.c](#).

Figure 3: Dokumentasjon for `int memory_reverse_copy`.

◆ memory_multiply_elements()

```
void memory_multiply_elements ( int * p_buffer,  
                               int  factor,  
                               int  size  
                               )
```

Multiply all the elements in p_buffer, of size size with the supplied factor.

Parameters

[in, out] **p_buffer** Buffer of integers to be multiplied with factor.
[in] **factor** Factor to multiply each of the elements in p_buffer with.
[in] **size** Size of p_buffer.

Warning

If p_buffer is NULL, the function will abruptly terminate the program with exit code 1.

Definition at line 20 of file [memory_library.c](#).

Figure 4: Dokumentasjon for `void memory_multiply_elements`.

2 Oppgave

Dere har fått utlever en mappe som heter “`document_this`”. Her ligger det to enkle biblioteker; “`person.h`” og “`substances.h`”, som begge har litt slapp dokumentasjon. Deres oppgave er:

1. Opprett en konfigurasjonsfil for Doxygen.
2. Les gjennom implementasjonsfilene (`.c`) for å skjønne hva som foregår.
3. Dokumenter headerfilene (`.h`) på en fornuftig måte.
4. Kjør Doxygen for å generere HTML.

Når dere er ferdige, viser dere en nettside med dokumentasjon til en studass for godkjenning.