C kodestandard

Søren Nørgaard

Sidst opdateret: 27. maj 2013

Dette dokumenterer en kodestil, der med fordel kan anvendes i sproget C. Hvis ikke andre kodestile er krævet, kan denne med fordel følges, for at skabe ens udseende kode i projektet. At anvende ens kodestil i et projekt gør koden lettere at læse, da alting er at finde på samme plads.

Denne kodestil tager udgangspunkt i Kerninghan & Richie's kodestil (K&R), og har desuden hentet inspiration fra Linux-kernelen mm.

På de sidste sider er et kodeeksempel, hvor de fleste dele er eksemplificeret. (Koden gør absolut intet, men gør det måske lettere at forstå hvad der menes).

God kodning!

Indhold

at sætte et par editorer op.

Vim

I Vim gives ovennævnte effekt ved at placere følgende i opstartsfilen `~/.vimrc`:

```
set encoding=utf-8
set tabstop=4
set shiftwidth=4
set expandtab
set autoindent
set cinoptions=:0,l1,t0,g0,(0
syntax enable
```

Emacs

I Emacs indsættes følgende i opstartsfilen `/.emacs` eller `~/.emacs.d/init.el`:

```
(setq-default tab-width 4)
(setq-default c-basic-offset 4)
(setq-default indent-tabs-mode nil)

(prefer-coding-system       'utf-8)
(set-default-coding-systems 'utf-8)
(set-terminal-coding-system 'utf-8)
(set-keyboard-coding-system 'utf-8)
(setq default-buffer-file-coding-system 'utf-8)
(setq x-select-request-type '(UTF8_STRING
    COMPOUND_TEXT TEXT STRING))
```

2 Navngivning

- Alle navne skrives på engelsk.
- Globale funktioner (der ikke er `static`), bør have sigende, unikke navne som `spi_send_byte()`.
- Funktioner der er static (kun kan ses i den kildefil de er i), bør have korte navne, der er sigende indenfor filens eget område (eks. `startclk()`)
- Samme gælder variable: Globale variable skal have lange/sigende navne, mens lokale variable bør holdes korte, og sigende indenfor sin givne funktion.
- Funktions- og variabelnavne skrives kun med små bogstaver, og ord er separerede med _.
- Ofte brugte variabelnavne er:
  - `i, j, k` til indeksering.
  - `p, q` til pointere.
  - `c` til karakterer.
  - `s` til string.
  - `x, y, z` til floats/doubles.
  - `l` til long.
  - `n` til ints.
- Ved "modsatte" funktioner benyttes modsat navngivning. Eksempeltvis: `setclk()` og `getsck()`, `readbyte()` og `writebyte()` osv.
- Defines skrives med `BLOKBOGSTAVER`, og makroer ligeså. Eks. `#define MAX(A,B) a>b?a:b`.

  Hvis makroer opfører sig som funktioner, skrives de som funktioner. Eks. `#define udelay(us) _delay_us(us)`.
- Fejlkoder defineres som negative tal, er sigende for koden den beskriver, og indeholder et `E` for error. Eks: `#define SPI_ENACK` for en fejl ved et spi-modul, der bliver NACK'et.

3 Kommentarer og kodedokumentation

3.1 Dokumenterende kommentarer

- Før hver globale funktion, indsættes en blokkommentar, der starter med `/**`, og har en * indenteret med 1 space for hver linje. Blokken afsluttes med en tom linje med `*/`.
- Blokken startes med en kort beskrivelse af hvad funktionen gør.
- For hver parameter indsættes en linje begyndende med `@param x Beskrivelse af x.`.
- For returværdier indsættes en linje som: `@return 0 = success, SPI_ENACK = No acknowledge.`, hvor alle returværdiers betydning dokumenteres.
- Dokumenterende kommentarer til funktioner forefindes ved funktionernes prototyper (i `.h`-filer).
- static funktioner (i `.c`-filer) kan med fordel indeholde en kommentar over funktionen, der forklarer returværdier, men bør ellers være sigende i sig selv.
- Dokumentation skrives på engelsk.

3.2 Almindelige kommentarer

- Forsøg at lave din kode så forståelig som muligt uden kommentarer, og brug dem kun hvor det er nødvendigt.
- Skriv ikke unødvendige kommentarer ("`i++; /* i is incremented by 1. */`").

- Kommenter så vidt muligt for enden af en linje, så koden kan læses uforstyrret. Hvis en stor kommentar skal knyttes, skriv den da som en blokkommentar som her:

  ```
  /* FIXME:
   * Some work needs to be done, to let
   * the user know an ACK has occured. */
  ```

  med en tom linje under og over, så kommentaren står for sig selv.
- Kommentarer skrives på engelsk, og skrives som om man skriver til kode. Eks: `/* Find the student with the best grades. */`.

4 Filopdeling

Skrives eksempeltvis et main-program, filen `main.c`, med et bibliotek bestående af `{spi.h, spi.c}`, opdeles filerne som følger.

Filer kodes i øvrigt *altid* i UTF-8-format.

4.1 main.c

- Inkluderer `spi.h`.
- Benytter funktioner herfra.

4.2 spi.h

- Indeholder prototyper til globale funktioner.
- Indeholder tilhørende dokumentation af disse funktioner.
- Indeholder `typedef`s, `struct`s og `#define`s tilhørende biblioteket.

4.3 spi.c

- Inkluderer `spi.h`.
- Indeholder, øverst, statiske funktioner, der benyttes senere i globale funktioner.[1]
- Indeholder, nederst, kode til de globale funktioner, der er defineret i `spi.h`.

[1] `static`-funktioner benyttes for at dele koden om i simplere underfunktioner. "End funktion skal gøre 1 ting, og gøre det godt!". Bare tænk på $\sin(x)$.

5 Kodeeksempel

5.1 spi.h

```
#include <stdio.h>

#ifndef SPI_H
#define SPI_H
#define RED 0x00FF0000
#define BLUE 0x0000FF00
#define GREEN 0x000000FF

#define MAX(A,B) a>b?a:b
#define udelay(us) _delay_us(us)

typedef struct {
    unsigned char mosi;
    unsigned char miso;
    unsigned char sck;
} spi_adapter;

/**
 * Sends one byte via SPI.
 * @param byte Data byte to be sent.
 * @param divide Number to divide the clock by.
 * @param slave ID of the slave to send to.
 *
 * @return 0 = success, SPI_ENACK = No acknowledge.
 */
int spi_send_byte(unsigned char byte, unsigned byte divide,
                  unsigned char slave);

#endif
```

5.2 spi.c

```
#include "spi.h"

static void startclk()
{
    SPICTL |= 0x01;
}

int spi_send_byte(unsigned char byte, unsigned byte divide,
                  unsigned char slave)
{
    printf("Så er den sendt...\n");

    if (SPICTL & 0x01)
        return 0;
    else
        return SPI_ENACK;
}
```

5.3 main.c

```
#include "spi.h"

int main()
{
    unsigned char input, c;

    input = getchar();
    switch (input) {
    case 0x01:
        spi_send_byte(0x01, 1, 0x01);
        break;

    case 0x02:
        c = spi_receive_byte();
        break;

    default:
        printf("Command unknown: %x\n", input);
        goto bailout;
    }

    /* Underneath are some different
     * ways to use if-else statements
     * for this standard */

    if (c > 100)
        printf("Det var et ordentligt tal!\n");
    else
        printf("Ja det fint...");

    if (c == 0x01) {                    /* Switch colors. */
        change_color(BLUE);
        printf("Skiftet til blå");
    } else if (c == 0x02) {
        change_color(RED);
        printf("Skiftet til rød");
    } else {
        change_color(green);
        printf("Skiftet til grøn");
    }

    if (c < 10 || input == 0x03 || input == 0x04
        || input == 0x05) {
        print_help();
        delay(1000);
    }

    return 0;

bailout:
    return -1;
}
```