SN C Coding Standard

Søren Nørgaard

22. 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.

1 Visuel organisering

Visuel opsætning af kode, herunder indentering og parentes-sætning (curly brackets).

1.1 Indentering

- Identering er 4 spaces.
- Labels, ifm. `goto`'s, indenteres ikke, og sættes helt til venstre.
- Labels ifm. `switch`'e indenteres på samme niveau som `switch`.

1.2 Curly brackets

- Curly brackets sættes på samme linje som udtrykket det tilhører (`if`, `while` etc.), pånær ved funktioner!
- Ved funktioner sættes curly brackets på linjen under funktionsudtrykket, som en linje for sig.
- Brug ikke curly brackets efter `if`, `else`, `for`, `while`, hvis kun en enkelt statement følger. Denne sættes på en linje for sig, og indenteres.

- I `if .. else if .. else`-strukturer, sættes `else if` og `else` på samme linje som tidligere niveaus afsluttende curly bracket.
- Efter sluttende curly bracket i en `do-while` konstruktion, sættes `while` på samme linje.

1.3 Whitespace

- Whitespace benyttes konsekvent til at opdele dele af koden, der ikke hører sammen – ligesom man ville gøre med afsnit i en bog.
- Der benyttes (som hovedregel) ikke mere end 1 tom linje til at opdele kode.

1.4 Linjelængde

- Linjer skal holdes under 80 karakterer. Dette er gammel praksis, men er meget smart hvis man vil printe kode i en rapport eksempeltvis.
- Hvis funktioners parametre fylder mere end 80 karakteres, kan linjen knækkes ved et komme, og alignes under funktionens begyndende parentes (under første parameter).
- Hvis andre strukturer `if`, `for` osv. bliver for lange, kan disse flyttes ned på næste linje, og indentes. Deles der ved operatorer kan disse med fordel placeres på den nye linje.
- Lange konstante strings, må gerne overstige 80 karakterer, for at gøre det lettere at søge på dem.

1.5 Editor-opsætning

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 karaktere.
  - `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 retur- værdiers betydning dokumenteres.
- Dokumenterende kommentarer til funktioner forefindes ved funktionernes prototyper (i `.h`-filer).
- static funktioner (i `.c`-filer) kan med fordel indeholde en kommentar, der forklarer returnværdier, men bør ellers være sigende i sig selv.
- Dokumentation skrives på engelsk.