Kodningsriktlinjer För Datatekniskt Projekt

Gabriel Käll

19 oktober 2020

Innehåll

1 Introduktion **1**

2 Referenser **1**

3 Riktlinjer - C **1**
3.1 Språk 1
3.2 Namngivning 1
3.3 Kommentarer 1
3.4 Whitespace 2
3.4.1 ”Måsvingar” 3
3.5 Filer 3
3.6 Deklarationer, uttryck, satser 3
3.6.1 Funktioner 3
3.7 Övrigt 4

1 Introduktion

Varför är kodningsriktlinjer nödvändiga?

Att ha en bra struktur är mycket viktigt när man kodar. Det hjälper till med att fånga fel innan dom blir problematiska, gör koden allmänt mer läsbar, och är en bra grund att falla tillbaka på och kolla igenom då programmet beter sig oväntat. Detta är ytterst viktigt i grupparbeten, både för att göra den slutgiltiga produkten mer sammanhängande, men också för att göra det lättare för gruppmedlemmarna att förstå varandras kod.

2 Referenser

I denna text kommer [1] användas som grundläggande källa för konventionerna. Det är en modifierad version av "Indian Hill C Style", enligt källan. I slutändan spelar det mest roll om man håller sig till en standard och är konsekvent, så det finns inte mycket att säga om varför denna standard skulle vara bättre.

3 Riktlinjer - C

3.1 Språk

Alla namn, kommentarer och annat språk som inte är kod ska skrivas på *Engelska*.

3.2 Namngivning

- Makrodefinitioner, konstanter, typedefs och Enum-konstanter skrivs med stora bokstäver. Ord separeras med '_'.
  Exempel:

```
#define MAX_POINTS 2
enum day{MON, TUE};
```

- Funktioner skrivs med små bokstäver och ord separeras med '_'.

```
int point_counter(void);
```

- Övriga variabelnamn skrivs med små bokstäver. Håll dessa korta men fortfarande beskrivande. Det är värt att fundera på vad variabler ska heta. Notering: Om dessa börjar bli långa och förkortning av namn blir otydligt, separera ord med '_'.

3.3 Kommentarer

- Kommentarer ska beskriva vad som händer och hur det görs (om det är oklart). Det är viktigt att kommentera gällande buggar / restriktioner. Undvik att skriva kommentarer som inte är relaterade till koden i närheten.

- Små kommentarer: Undvik att kommentera på för låg nivå, d.v.s. att kommentera som man gör i Assembler. (en kommentar på varje line om vad koden gör) Förklara *vad* som händer, inte *hur* det händer.

  Exempel: Istället för att skriva ”sum of numbers divided by n”, skriv ”compute mean”.

- Kommentarer som beskriver klasser, algoritmer, datastrukturer ska vara *blockkommentarer*.
  Exempel:

```
/*
 * Block comment
 *
 */
```

- Om t.ex. funktioner behöver grupperas, använd kommentarsstandarden som visas i 3.4.

3.4 Whitespace

- Inuti ett block ska allt indenteras 1 tab in. Exempel:

```
if (1) {
    h = 0;
}
```

- Det ska vara 1 blank line mellan funktioner. Om du har separerat funktionerna i olika grupper ska det vara 4 lines mellan slutet av förra gruppen till den nya. Exempel:

```
/*General object funcs*/
POBJECT obj_init(PGEOMETRY g, int x, int y,int dirx, int diry) {
    ...
}

void obj_deinit(POBJECT p) {
    free(p);
}




/*Collisions*/
int objects_overlap(POBJECT o1, POBJECT o2) {
    int x  = o1 -> posx;
    int y  = o1 -> posy;
..
```

3.4.1 ”Måsvingar”

– Måsvingar som används i samband med ”block” ska skrivas såhär:

```
void function(int i) {
...
}

if (1) {

}
```

3.5 Filer

- C-filerna ska separeras i olika kategorier eller delas upp efter komponenter. T.ex. ska I/O-relaterade funktioner ha en separat ”.c” fil.
- Så kallade ”include guards” bör användas längst upp alla headerfiler för att undvika att dom blir inkluderade 2 gånger.
  Exempel:

```
#pragma once
```

- Om det behövs (om en headerfil blir orimligt lång) ska flera headerfiler användas, separerade i olika kategorier.

3.6 Deklarationer, uttryck, satser

- Pekardefintionen ’*’ ska skrivas brevid variabelnamnet.

```
//Wrong, j and k does not init as pointers
char* i, j, k;
//Correct
char *i, *j, *k;
```

- Om ett uttryck känns svårläst, dela upp det i flera rader:

```
if (x1 < x || y1 < y
 && x1 > x && y1 > y) {
 }
```

3.6.1 Funktioner

- En blockkommentar ska placeras innan varje större funktion för att beskriva vad den gör och beskriva invariabeln samt utvariabeln. Om det inte är enkelt att se hur man använder den, förklara det med.

```
/*
 * Calculates a...
 * Usage:
 * Input:
 * Output:
```

```
*/
void function(int i){
...
}
```

- ”Måsvingar” innan och efter funktioner ska placeras som det beskrivs i 3.4.1.

3.7 Övrigt

Här läggs mer standarder upp om det behövs.

Referenser

[1] M. B. Henry Spencer, David Keppel. (2008) Reccomended c style and coding standards. [Online]. Available: https://www.doc.ic.ac.uk/lab/cplus/cstyle.htmlN10081