
Dit document bevat richtlijnen voor het programmeren bij het eerstejaars college *Programmeer-methoden*, Universiteit Leiden, najaar 2010, zie

www.liacs.nl/home/kosters/pm/

Met dank aan allen die aan deze tekst hebben bijgedragen.
Walter A. Kosters, Leiden, 23 augustus 2010.

Richtlijnen bij programmeeropgaven

Een belangrijke taak van de programmeur is ervoor te zorgen dat zijn of haar code begrijpelijk is. Bij Programmeermethoden moet je code tenslotte nagekeken worden en ook later, bijvoorbeeld bij een bedrijf, heb je meestal collega's die willen begrijpen wat jij gedaan hebt. En later wil je je eigen code ook nog steeds snappen. Verschillende dingen kunnen ervoor zorgen dat code begrijpelijk is: een nette layout, goed commentaar en een goede modulariteit. Wat voor richtlijnen wij hierbij hanteren, is hieronder kort toegelicht. Uiteraard is over deze regels discussie mogelijk.

1 Commentaar

- In principe moet op alle gebruikte variabelen commentaar gegeven worden. Alleen als het commentaar triviaal zou zijn, mag het achterwege gelaten worden. Een voorbeeld van noodzakelijk commentaar is:

```
int i; // teller voor het aantal pogingen
```

Overbodig commentaar is:

```
int pogingteller; // teller voor het aantal pogingen
```

Zoals uit dit voorbeeld blijkt, is commentaar uit te sparen door duidelijke namen voor variabelen te kiezen. Door duidelijke namen te gebruiken wordt de verdere C++ code bovendien ook veel leesbaarder.

- Bij elke functie moet commentaar staan dat de bedoeling van de functie toelicht. De betekenis van elke parameter moet uitgelegd worden. Probeer een zin (of misschien twee) te maken die de werking beschrijft in termen van de parameters. Geef zoveel mogelijk aan waarom parameters een bepaald type hebben.
- Bovenaan elk C++ bestand moet commentaar staan. Dit commentaar moet ten minste bevatten:
 - de naam van de auteur(s);
 - de naam van het bestand;

- wat het programma(onderdeel) doet;
- welke compiler(s) en programma(s) gebruikt zijn om het bestand te maken en te testen;
- de datum waarop (voor het laatst) aan het bestand gewerkt is.

2 Layout

- Maak de regels niet te lang. Een maximum van 70 karakters per regel zorgt er voor dat op iedere printer en op (bijna) ieder beeldscherm de regels niet afgebroken worden.
- De declaraties van variabelen staan bovenaan de functie(s); main is ook een functie!
- Alle instructies die *binnen* een if, while, else, do, for, switch of class vallen, alsmede alle instructies die binnen functies staan, moeten ingesprongen worden ten opzichte van die if, while, etc. als ze bij die die if, while, etc. horen. De volgende layout is fout:

```
if ( k == 0 )
k++;
```

De volgende layouts zijn acceptabel:

```
if ( k == 0 )
    k++;
else if ( j == 0 )
    j++;
```

of

```
if ( k == 0 )
    k++;
else
    if ( j == 0 )
        j++;
```

- Bij functies zijn verschillende mogelijkheden:

```
int doewat (int x) {
    return x-1;
} // doewat
```

of

```
int doewat (int x)
{
    return x-1;
} // doewat
```

- Een regel bevat niet meer dan twee instructies, en doorgaans niet meer dan één.
- De layout moet consequent zijn, bijvoorbeeld:
 - waar ingesprongen wordt, moet overal evenveel ingesprongen worden. Fout is:

```
if ( k == 0 )
    k++;
if ( l == 1 )
    l++;
```

De laatste regel is namelijk iets meer ingesprongen dan de tweede regel.

- als in een situatie ingesprongen wordt, moet bij voorkeur in alle “vergelijkbare” situaties ook ingesprongen worden. “Vergelijkbaar” moet niet al te streng opgevat worden. Ter illustratie, de volgende layout is niet erg mooi omdat in “vergelijkbare” situaties niet op dezelfde manier ingesprongen wordt:

```
if ( k == 0 )
    k++;
if ( l == 1 ) k++;
```

Enkele “overtredingen” van dit criterium worden meestal getolereerd. Te zien is overigens dat de variabele `l` lastig van de `1` te onderscheiden is!

- Accolades staan bij voorkeur altijd op “dezelfde” plaats. Niet mooi, want inconsequent, is:

```
if ( k == 0 ){
    k++;}
if ( l == 1 )
{ l++;
}
```

Ook hier worden enkele “overtredingen” meestal getolereerd.

3 Modulariteit

- Er moeten zo min mogelijk globale variabelen gebruikt worden, liefst 0.
- Gebruik zoveel mogelijk geschikte functies.
- Maak de functies niet te lang: ze zouden op een klein beeldscherm (in één klein window) goed te volgen moeten zijn.
- Splits grote programma’s in meerdere files.

4 Conclusie

Ten overvloede tot besluit nog: deze richtlijnen zijn er niet om studenten te pesten! Kortom: ze zijn er voor ieders bestwil. Een klein voorbeeld tot besluit. De fout in de volgende stukjes code is niet direct te zien door de slechte layout:

```
while ( k < 10 )  
cout << k;  
k++;
```

```
while ( k < 10 )  
    cout << k;  
    k++;
```