

```
=====  
Indentation (Sourcecodeformatierung) mit vim  
=====
```

Vim unterstützt automatische Einrückung. Dafür empfehlen sich folgende Einstellungen (einfach in die Datei ~/.vimrc einsetzen):

```
set sts=4 ts=8 sw=4
```

Im Einzelnen

```
ts=8      stellt die Breite eines Tabs ein (nichts anderes als 8 einstellen)
sts=4     stellt ein, wieviele Spaces vim automatisch vorrückt, wenn man auf
          die Tabtaste drückt
sw=4     stellt für vims eingebauten Indentationalgorithmus ein, wieviele
          innerhalb eines Blocks (also { ... }) eingerückt werden soll
```

Um Teile des Sourcecodes umzuindenten empfiehlt sich

```
v        fängt mit einem neuen visuellen Block an
bewegen  bis zum Ende des Blocks der neu eingerückt werden soll
=        eigentliches Einrücken
```

Insbesondere praktisch ist es, wenn man auf einer öffnenden oder schliessenden geschweiften Klammer steht ("{" oder "}"), die Taste % zu verwenden, um zur passenden Klammer zu gehen. Die gesamte Einrückung kann dann mit 3 Tasten vorgenommen werden:

```
v%=      rückt Block zwischen passenden Klammern neu ein
```

Diese spezielle Operation geht sogar noch kürzer mit

```
=%       rückt Block zwischen passenden Klammern neu ein
```

Man kann vims Einrückung sehr viel genauer abstimmen, es gibt viele andere Optionen.

```
:help C-Indenting
```

Zwei weitere Befehle sind nach-rechts Einrücken und nach-links Einrücken:

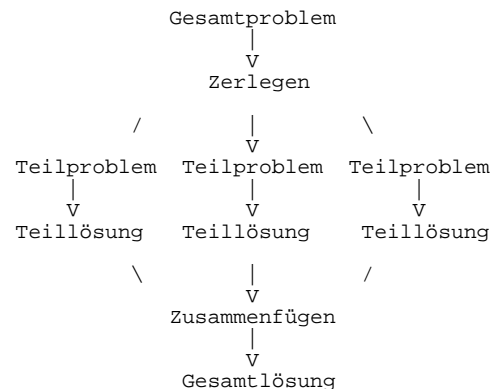
```
v        fängt mit einem neuen visuellen Block an
bewegen  bis zum Ende des Blocks der neu eingerückt werden soll
< >     eigentliches Einrücken
```

die sich wieder analog verkürzen lassen zu

```
v%>     erhöht Indentationlevel zwischen passenden Klammern
v%<     erniedrigt Indentationlevel zwischen passenden Klammern
>%      erhöht Indentationlevel zwischen passenden Klammern
<%      erniedrigt Indentationlevel zwischen passenden Klammern
```

```
=====  
Divide and Conquer Algorithmen  
=====
```

Divide and Conquer Algorithmen sind solche, die das Problem in einfachere Teilprobleme zerlegen, und diese Teilprobleme isoliert lösen, und dann die Lösungen zusammenfügen. Es ergibt sich also das Schema für einen Schritt:



Dies lohnt sich natürlich nur, wenn das Zerlegen, das Lösen der Teilprobleme und das Zusammenfügen der Lösungen insgesamt weniger Aufwand bedeutet als das Lösen des Gesamtproblems.

Oft lässt sich der Zerlegungsschritt auch auf die einzelnen Teilprobleme wieder anwenden, sodass man das Problem solange immer weiter zerlegt, bis die Lösung sehr einfach wird oder sich das Problem nicht weiter zerlegen lässt.

