

8. Dezember

Stepwise Refinement : am Beispiel der Bonus Exercise 2.

1. Input (gegeben)

2. Backtracking (solve-from)

2.1 nächstes Feld

> falls letzte col :
row+1, col=0

> sonst
row, col+1

> Beginne beim ersten Feld, setze #

> Kann man nun die restlichen Felder so füllen, dass alle Hints stimmen? \Rightarrow Rekursion

Pseudo-Code:

gebe # ein

if solve-from (nächstes Feld)
return true

> falls nicht, fülle mit "." aus

gebe "." ein

if solve-from (nächstes Feld)
return true

> sonst kann das Nongramm nicht gelöst werden

else

return false

> falls das letzte Feld erreicht ist, wird gecheckt, ob Lösung richtig ist

if last row and last col

return check-solution

3. (gehört eig zu 2) Checken, ob grid richtig ausgefüllt ist (check-solution)

> checke zuerst Reihen

→ wo erster #

→ zähle bis wieder "."

→ vergleiche mit hint

→ wiederhole so oft, wie Anzahl hints

→ gibt es weiteren # \Rightarrow false

→ ansonsten nächste Reihe

> cols : genau gleich !

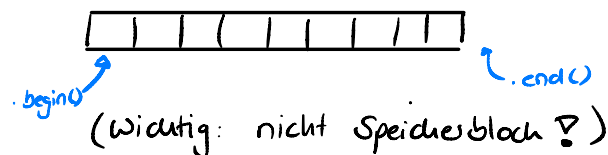
4. Output des grids (gegeben)

iterator : Pointer ermöglichen das Durchlaufen zusammenhängender Speicherblöcke. Wir können sie jedoch nicht für das Durchlaufen von Containers (Vektoren, strings, Sets etc) verwenden, da wir nicht wissen, wie diese implementiert sind. Deshalb haben Containers **iterators** die genau dies ermöglichen.

Def : `std::Containestyp::iterator it;`

Iterator auf .at 0 \Rightarrow `it = container.begin()`

Iterator auf nach container \Rightarrow `it = container.end()`



const

auf Objekt : `std::vector::const_iterator ...`

auf Iterator : `const std::vector::iterator ...`

set : geordneter Vektor ohne Duplikate.

! Achtung: kein `[]`, `+`, `-`, `<`, `>`, `<=`, `>=`, `+=`, `-=`

aber `++`, `--`, `==`, `!=`

\Rightarrow gut um zu überprüfen, ob Element enthalten