Thema, Ziele: Code Bloat bei Templates

Aufgabe 1: Vermeiden von Code Bloat bei Templates, Code Hoisting

Alle Programme wurden mit dem Compiler g++ v3.4.4, bzw. g++ v4.5.3 übersetzt.

a) just do it. Beachten Sie, dass einzelne Elementfunktionen implizit inline sind.

b) Wenn ohne Optimierung compiliert wird, sind keine Elementfunktionen inline. Alle Elementfunktionen sind einfach vorhanden und werden mit `call` aufgerufen.
Wenn mit Optimierungstufe 3 compiliert wird, sind bei g++ v3.4.4 die impliziten inline-Elementfunktionen inline. Alle weiteren Elementfunktionen sind einfach vorhanden und werden mit `call` aufgerufen. Bei g++ v4.5.3 sind sämtliche Funktionen der Klasse `Stack` inline.

Optimierung O…	g++ v3.4.4	g++ v4.5.3
0	482'188	55'914
3	481'177	55'003

Wie aus einem früheren Praktikum bekannt ist, stammen bei der g++-Version v3.4.4 mit der 3. Optimierungsstufe 469'345 Bytes von der Bibliothek `iostream`., d.h. der Nettocode bei Optimierungsstufe 3 beträgt 11'832 Bytes.

c) siehe `./Loesung/A1_1`

Wenn ohne Optimierung compiliert wird, sind bei beiden Compilern keine Elementfunktionen inline. Alle Elementfunktionen inklusive die Konstruktoren sind doppelt mit praktisch identischem Code vorhanden und werden mit `call` aufgerufen. Die Texte, die mit `cout` ausgegeben werden, sind nur einfach vorhanden.
Wenn bei g++-Version v3.4.4 mit Optimierungstufe 3 compiliert wird, sind die impliziten inline-Elementfunktionen inline. Alle weiteren Elementfunktionen sind ebenfalls doppelt vorhanden und werden mit `call` aufgerufen.
Wenn hingegen g++ v4.5.3 genommen wird, sind alle Elementfunktionen ausser `StackUI::dialog()` inline. Diese Funktion ist dann zweifach vorhanden, je für einen Stack der Grösse 3 und der Grösse 4, und wird wie folgt aufgerufen:

```
call   __ZN7StackUIIiLi3EE6dialogEv
...
call   __ZN7StackUIIiLi4EE6dialogEv
```

In der folgenden Tabelle sind die Codegrössen ersichtlich.

Optimierung O…	g++ v3.4.4	g++ v4.5.3
0	483'705	58'589
3	482'351	56'278

d) Code hoisting, siehe `./Loesung/A1_2`. Beachten Sie auch die Kommentare im Source Code.

In der Klasse `Stack` sollen alle von der Grösse unabhängigen Teile in eine Basisklasse ausgelagert werden. Die Analyse, was unabhängig von der Grösse ist, ist nicht so trivial. Sind beispielsweise die Elementfunktionen `isEmpty()` und `isFull()` unabhängig von der Grösse oder nicht? Weiss man das bereits bei der Deklaration oder ist es erst bei der Implementation bekannt? Aus der Implementation geht hervor, dass `isEmpty()` unabhängig ist, `isFull()` jedoch nicht.
Alle weiteren Elementfunktionen, die `isFull()` benötigen, sind demnach auch nicht unabhängig, das ist z.B. `push()`.
Damit die Unterklassen einfach auf die Attribute zugreifen können, sollten `top` und `error` als `protected` statt `private` definiert werden. In der Basisklasse ist der Array `elems[]` nicht bekannt, da dieser ja eben abhängig von der Grösse ist. Also müssten beinahe alle Elementfunktionen (alle, die auf den Array zugreifen) in die Unterklasse. Um das zu verhindern kann im Ctor der Basisklasse ein Pointer auf den

`elems`-Array gesetzt werden. In der Basisklasse wird dann damit gearbeitet. Dies ist zwar nicht gerade schön, allerdings löst es den Code bloat.

Wenn ohne Optimierung compiliert wird, sind keine Elementfunktionen inline. Alle Elementfunktionen und Konstruktoren der Basisklasse sind nur noch einfach vorhanden, alle anderen Elementfunktionen sind doppelt vorhanden.
Wenn mit Optimierungstufe 3 compiliert wird, sind die impliziten inline-Elementfunktionen inline. Funktionen der Basisklasse sind nur noch einfach, alle anderen Funktionen doppelt vorhanden.

Optimierung O…	Codegrösse [Bytes]	Delta zur Vorgabe [Bytes]	Delta zu A1_1
0	483'386	1'198	-319
3	482'256	1'079	-95