AUFGABE 4: ENTWURF UND TEST VON SOFTWARE

Legen Sie die vorherige Aufgabe in den Unterordner `01_git` und übertragen Sie die Änderung in Ihr git-Repository. Laden Sie die aktuelle Vorgabe mittels git herunter. Hierfür müssen sie zunächst ein neues entfernte Vorgabe-Repository einbinden und die Vorgabe für diese Aufabe herunterladen:

```
git remote add vorgabe gitosis@i4git:vezs_ss15_vorgabe
              git pull vorgabe master
```

In dieser Aufgabe werden Sie einen abstrakten Datentyp zur Verwaltung einer *Prioritätswarteschlange* implementieren und testen. Sie können hier davon ausgehen, dass Sie nicht den Einschränkungen eines eingebetteten Systems unterliegen – d. h. die Verwendung von `malloc` und `free` ist zulässig und Sie können die Details Ihrer Datentypen vollständig vor dem Anwender verbergen. Wenden Sie die *Objektbasierte Entwurfsmethode*, die Sie in einer der vorangegangenen Tafelübungen kennengelernt haben, an. Anschließend werden Sie Ihre Warteschlangenimplementierung mit Hilfe feingranularer Testfälle testen. Dabei soll mindestens vollständige Codeüberdeckung erreicht werden. ☞ lcov

Anforderungsdokument

Die Warteschlange soll folgenden Anforderungen genügen: Es soll möglich sein eine beliebige Anzahl von Warteschlangen zu erstellen und die maximale Anzahl der Elemente einer solchen Warteschlange soll nur durch den verfügbaren freien Speicher und die Zahl der Prioritätsebenen begrenzt sein. Jede Prioritätsebene soll innerhalb einer Warteschlange genau einem Element gleichzeitig zugewiesen sein. Die Warteschlange soll nicht-vorzeichenbehaftete 64 Bit-Ganzzahlen verwalten.

Versucht der Benutzer ein Element mit einer Priorität einzufügen, die in der Warteschlange bereits vergeben ist, so ist dies ein Fehler. Dieser soll nicht durch eine Ausgabe auf dem Bildschirm sondern durch einen Rückgabewert angezeigt werden. Geht während der Programmausführung der Hauptspeicher aus, so ist dies ein Fehler, über den

der Benutzer über die API benachrichtigt wird. Die Reaktion auf den Fehler ist dem Aufrufer überlassen.

Die Warteschlange soll die Möglichkeit bieten, das höchstpriore Element zu entfernen und hierbei den Wert dieses Elements dem Benutzer zugänglich machen. Außerdem soll es möglich sein über die Elemente in der Warteschlange zu iterieren – der Iterator soll beim höchstprioren Element anfangen und die Elemente in Reihenfolge absteigender Priorität besuchen. Ferner soll es möglich sein, die Anzahl Elemente, die sich derzeit in der Warteschlange befinden zu erfragen.

Im Fehlerfall soll sich die Warteschlange gutmütig verhalten, d. h. ein Speicherzugriffsfehler oder nicht definiertes Verhalten sind beispielsweise nicht akzeptabel. Sofern das Verhalten der Warteschlange vom Laufzeitsystem abhängig ist, soll dies dokumentiert werden.

Aufgabenstellung

1. *Arbeitumgebung:* Initialisieren Sie die Vorgabe wie in den Ubungsfolien besprochen. Machen Sie sich mit den Konfigurations und den generierten Dateien vertraut.

☞ *cd* `build`
☞ `cmake ..`

2. *Softwareentwurf:* Entwerfen Sie mit Hilfe der *Objektbasierten Entwurfsmethode* Datentypen, die die geforderte Funktionalität bieten. Dokumentieren Sie dabei das Verhalten der resultierenden API.

3. *Testfallentwurf:* Entwerfen Sie nun Testfälle, die geeignet sind zu zeigen, dass die Implementierung Ihres Softwareentwurfs den Anforderungen entspricht und auch sonst kein nicht definiertes Verhalten besitzt. Ziel ist u. A. mindestens vollständige Codeüberdeckung zu erreichen. *Welche Randfälle müssen Sie prüfen und warum? Dokumentieren Sie diese!*

4. *Implementierung:* Setzen Sie nun den Softwareentwurf und den Testfallentwurf um. Hierbei sollten die Testfälle für eine Funktion möglichst nicht von der gleichen Person geschrieben werden, die die Funktion implementiert (Stichwort *Betriebsblindheit*). Überlegen Sie sich vor der Implementierung welche Teile der Implementierung voneinander unabhängig sind und teilen Sie den Teammitgliedern entsprechende Arbeitspakete zu.

Für die Implementierung der Testfälle sollen Sie die Testumgebung von `cmake` verwenden. Legen Sie hierbei pro Testfall eine eigene `.c`-Datei

mit `main()`-Funktion im Unterverzeichnis `tests` an und überprüfen Sie die erreichte Überdeckung mit Hilfe von `lcov`. Lassen Sie Ihre Testfälle auch mit aktivierten `clang`-Sanitizer laufen und korrigieren Sie ggf. autretende Fehler.
Hinweis: Es bietet sich an, die Warteschlange als einfach verkettete Liste zu implementieren.

☞ `make test`
☞ `make lcov`

Hinweise

- Bearbeitung: Gruppe mit je zwei bis drei Teilnehmern.
- Abgabezeit: die Woche vom 12.05.2015 bis zum 18.05.2015
- Fragen bitte an `i4ezs@lists.informatik.uni-erlangen.de`