AUFGABE 5: SOFTWARE-ENTWURF UND -TEST

In dieser Aufgabe werden Sie einen abstrakten Datentyp zur Verwaltung einer *Prioritätswarteschlange* implementieren und testen. Sie können hierbei 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* an, die Sie in der vorangegangenen Tafelübung U3 kennengelernt haben. Anschließend werden Sie Ihre Warteschlangenimplementierung mit Hilfe feingranularer Testfälle testen. Dabei soll mindestens vollständige Codeüberdeckung erreicht werden.

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.

Aufgaben

1. Arbeitsumgebung: Initialisieren Sie die Vorgabe wie in den Übungsfolien besprochen. Machen Sie sich mit der Konfiguration, den vorgegebenen Dateien 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.* Beachten Sie auch das in Tafelübung U3 vorgestellte Vorgehen zur Hauptwort- und Verbenextraktion.

3. Fehlerraumanalyse: Skizzieren Sie die Fehlerhypothese. *Welche Randfälle müssen Sie berücksichtigen und warum?*

. . . . . .
. . . . . .
. . . . . .
. . . . . .

4. Testfallentwurf: Entwerfen Sie nun auf Basis Ihrer Analyse und der identifizierten Randbereiche Testfälle, die geeignet sind zu zeigen, dass die Implementierung Ihres Softwareentwurfs den Anforderungen entspricht und auch sonst kein nicht definiertes Verhalten besitzt. *Welche Kategorien von Testfällen gibt es?*

. . . . . .

In welche dieser Kategorien fallen die einzelnen von Ihnen entworfenen Testfälle?

. . . . . .

Implementierung

5. Aufteilung: 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. *Wieso ist diese Herangehensweise sinnvoll?*

. . . . . .
. . . . . .

Überlegen Sie sich vor der Implementierung welche Teile des Anforderungsdokuments unabhängig voneinander zu entwickeln sind und teilen Sie den Teammit-

gliedern entsprechende Arbeitspakete zu. Achten Sie auch auf eine Trennung von Testfall- und Funktionsentwicklung. *Dokumentieren Sie die Arbeitsaufteilung:*

.....

6. Testen 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`. Beachten Sie, dass `make lcov` erst erfolgreich aufgerufen werden kann, falls schon mindestens ein Testfall ausgeführt wurde, der Ihre Prioritätswarteschlange auch tatsächlich verwendet. *Welche Anforderungen sind an gute Testfälle zu stellen?*

☞ `make test`
☞ `make lcov`

.....
.....
.....
.....

Welches Überdeckungskriterium überprüft `lcov`*?*

.....

Schreiben Sie mindestens so viele Testfälle, dass `lcov` eine 100%ige Überdeckung anzeigt.

7. `AddressSanitizer` Lassen Sie Ihre Testfälle auch mit aktiviertem `AddressSanitizer` laufen und korrigieren Sie ggf. auftretende Fehler. *Welche Fehler haben Sie mit Hilfe des* `AddressSanitizer`*s gefunden, die ohne den* `AddressSanitizer` *nicht zu einem Fehlverhalten führen?*

.....
.....

Diese Fehler scheinen somit ja keine Auswirkung zu haben. *Wieso ist der Einsatz des* `AddressSanitizer`*s trotzdem sinnvoll?*

.....
.....
.....

8. `Clang Static Analyzer` Testen Sie Ihr Programm mit dem `Clang Static Analyzer`. Dokumentieren Sie die Ergebnisse. *Wo liegen die Vor- und Nachteile dieses Ansatzes vor allem in Hinblick auf den* `AddressSanitizer`*?*

.....
.....

9. Nachbereitung: *Welche Aspekte Ihres ursprünglichen Entwurfs haben sich im Verlauf der Implementierung als unzureichend erwiesen?*

.....
.....
.....
.....

Wie haben Sie diese Probleme behoben?

.....
.....
.....

Hinweis: Es bietet sich an, die Warteschlange als einfach verkettete Liste zu implementieren.

Hinweise

- Bearbeitung: Gruppe mit je zwei bis drei Teilnehmern.
- Abgabezeit: 20.06.2016
- Fragen bitte an `i4ezs@lists.cs.fau.de`