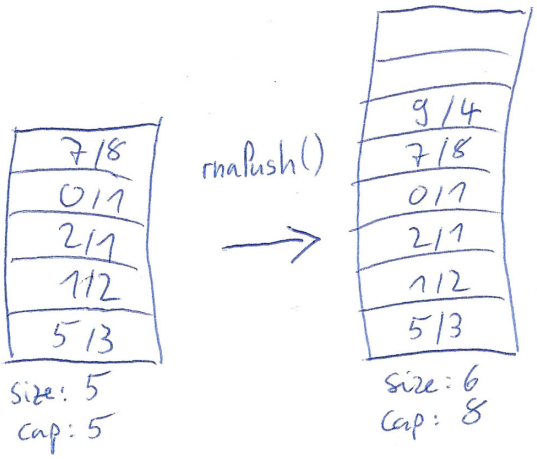
C++ für Java-Programmierer / Aufgabe 1.2



Übung 1 besteht aus zwei aufeinander aufbauenden Teilaufgaben und kann/soll über mehrere Wochen bearbeitet werden (siehe *Abgabe*). Dieses Blatt enthält Teilaufgabe 1.2 und baut auf Teilaufgabe 1.1 auf.

Aufgabe 1.2: `RationalNumberArray` in C

Implementieren Sie ein Modul (`.h` + `.cpp`) für den Datentyp `RationalNumberArray`. Dieser Datentyp repräsentiert ein dynamisches Array rationaler Zahlen vom Typ `RationalNumber` (siehe Aufgabe 1.1) im Hauptspeicher. Einem Benutzer soll es einfach möglich sein, weitere Brüche zu dem Array hinzuzufügen, auf Brüche an beliebiger Stelle im Array schreibend und lesend zuzugreifen und eine Reihe aufeinanderfolgender Brüche aus dem Array zu löschen.

- Ihr struct `RationalNumberArray` soll zunächst über die folgenden Attribute verfügen:

`RationalNumber *data`	Ein Zeiger auf ein Array von Werten des Typs `RationalNumber`
`int size`	Wie viele Elemente sind tatsächlich vom Benutzer in dem Array gespeichert worden?
`int capacity`	Für wie viele Elemente ist in dem Array zur Zeit Speicher alloziert?

- Implementieren Sie Ihren Datentyp verdeckt. Das bedeutet:
  - Der Datentyp ist in der Implementierungsdatei definiert.
  - In der Headerdatei befindet sich nur eine Deklaration des Typs.
  - Alle in der Headerdatei deklarierten Funktionen operieren niemals direkt auf einem `RationalNumberArray`, sondern immer nur auf Zeigern auf ein solches Objekt.

- Das `RationalNumberArray` (kurz: RNA) sollte über die folgenden Grundfunktionen verfügen:
  - **`rnaCreate()`** erzeugt ein neues RNA und erhält optional eine Größenangabe, wie viele Speicherplatz initial reserviert werden soll (z.B. Platz für 10 Brüche als Voreinstellung)
  - **`rnaDelete()`** gibt sämtlichen Speicher eines mittels **`rnaCreate()`** erzeugten RNA wieder frei
  - **`rnaResize()`** erlaubt es, ein existierendes RNA auf eine bestimmte Größe zu bringen, (egal ob kleiner oder größer)
  - **`rnaSize()`** und **`rnaCapacity()`** liefern die aktuelle Anzahl von Elementen und die aktuelle Speicherkapazität des Arrays zurück
  - **`rnaAdd()`** fügt einen Bruch als neues Element an das Array an. Falls das Array nicht mehr über genug Speicher verfügt, wird es automatisch vergrößert.
  - **`rnaSet()`** erlaubt es, ein Element an beliebiger Stelle in dem Array zu schreiben. Falls das Element bisher nicht existiert (also hinter dem bisher letzten Element liegt), wird das Array entsprechend vergrößert und alle noch nicht initialisierten Element mit dem Bruch-Nullwert {0,1} initialisiert.
  - **`rnaGet()`** liefert ein Element an beliebiger Stelle in dem Array zurück.
  - **`rnaRemove()`** löscht eine Folge von Elementen (von - bis) aus dem Array

- Implementieren Sie ein Testprogramm (in Ergänzung der Tests in `testRN.cpp`), welches die oben erwähnten Grundfunktionalitäten abdeckt. Achten Sie darauf, sinnvolle Tests zu definieren, die tatsächlich in der Lage sind, Fehler aufzudecken.

Für eine sehr gute Note sollten Sie zusätzlich folgendes umsetzen:

- Implementieren Sie Funktion `rnaError()`, die für ein bestimmtes RNA-Objekt abfragt, ob bei der letzten Operation ein Fehler aufgetreten ist (und welcher Art dieser Fehler war).
  - Als mögliche Fehler kommen u.a. in Betracht: ungültiges Array-Objekt, ungültiger Array-Index, kein Speicherplatz allozierbar, ... Definieren Sie zur Repräsentation dieser Fehlertypen ein entsprechendes `enum`.
  - Jede Operation auf einem RNA-Objekt sollte im Erfolgs- oder Fehlerfall den Fehlerstatus als Attribut in dem Objekt speichern. die `rnaError()`-Funktion liefert diesen Status zurück.

- Ergänzen Sie Ihr Testprogramm so, dass Sie `rnaError()` validieren können.

- Implementieren Sie einen Mechanismus, der bei einem auftretenden Fehler eine vom Benutzer spezifizierte Callback-Funktion aufruft. Diese Funktion soll mittels einer Funktion `rnaSetErrorCallback()` gesetzt werden können.

  - Der Benutzer sollte jedem `RationalNumberArray` eine eigene Fehlerfunktion zuordnen können. Dazu speichern Sie am besten einen Funktions-Zeiger direkt in dem RNA-Objekt.
  - Innerhalb der Fehlerfunktion sollte man auf die aktuellen Werte des betroffenen `RationalNumberArray` zugreifen können.

- Ergänzen Sie Ihr Testprogramm so, dass Sie den Callback-Mechanismus validieren können.

Weitere Hinweise für diese Aufgabe

- Verwenden Sie nur Sprachelemente der Sprache *C*, d.h. insbesondere keine Klassen, Methoden, Referenzen, Namespaces, Exceptions oder Templates. Das Interface sollte dennoch möglichst sauber sein, d.h. ein „Kunde" Ihres Codes sollte nur über die wohldefinierten Funktionen auf den Datentyp zugreifen müssen.
- Da der Datentyp RationalNumberArray potentiell umfangreich ist und verdeckt implementiert werden soll, übergeben Sie Objekte dieses Typs immer nur mittels Zeigern, niemals *by value*.
- Überlegen Sie sich eine vernünftige Vergrößerungs-Strategie für die Funktionen `rnaPushBack()` und `rnaSet()` - um wie viele weitere Werte soll die Größe des Arrays wachsen, wenn die aktuelle Kapazität erschöpft ist?

Abgabe und Demonstration

Die Abgabe der gesamten Aufgabe 1 soll bis zum 17.04.2012 (23:55 Uhr) in Moodle erfolgen. Verspätete Abgaben werden wie in den Handouts beschrieben mit einem Abschlag von 2/3-Note je angefangener Woche Verspätung belegt. Geben Sie bitte pro Gruppe jeweils nur eine einzige .zip-Datei mit dem Qt-Projektverzeichnis ab, welches die Quellen und die Projektdatei enthält - bitte keine Kompilate in der Abgabe!

Demonstrieren und erläutern Sie dem Übungsleiter Ihre Lösung in einer der Übungen, möglichst bis zum Abgabetag. Die Qualität Ihrer Demonstration ist, neben dem abgegebenen Code, ausschlaggebend für die Bewertung! Es wird erwartet, dass alle Mitglieder einer Gruppe anwesend sind und Fragen beantworten können.