Übungsblatt 6

Eine Bemerkung zu Tests für Ihre Software: Ich gehe bei allen Aufgabestellungen davon aus, dass Sie die von Ihnen entwickelten Funktionen und Klassen selbstverständlich compilieren und testen (dafür muss i.A. extra Code geschrieben werden!), auch wenn ich das nicht extra als Aufgabe in die Aufgabenstellung aufnehme!

1. Es sei die folgende Template-Funktion gegeben:

```
template<class S, class T, int var>
S f(T x, S y) {
    S z = y.compute(x);
    for (int i = 0; i < var; i++) {
        z = z.compute(y, z);
    }
    return z;
}
```

   (a) Geben Sie die Realisierung von `f` an, die durch `int, double, 5` entsteht! Ist sie korrekt? Falls nein: Warum nicht?

   (b) Schreiben Sie ein `C++`-Programm, das die obige Template-Funktion enthält. Fügen Sie dem Programm eine `main`-Funktion hinzu, in der eine durch `int, double, 5` realisierte Version von `f` mit den Argumenten `5` und `7.0` aufgerufen wird. Compilieren Sie das Programm, sehen Sie sich die Fehlermeldung an und erklären Sie sie!

   (c) Entwickeln Sie Datentypen mit denen man `f` korrekt realisieren kann. Schreiben Sie ein eigenes Programm, das diese Datentypen enthält, sowie einen mit diesen Typen parametrisierten Aufruf einer korrekten Realisierung von `f`!

   (d) Im Moodle finden Sie die Dateien `beautifulFunction.h` und `beautifulFunction.cpp`. Im Moodle finden Sie eine weitere Datei `Ue6A1main.cpp`. Erweitern Sie diese um den Aufruf der Realisierung von `f`, den Sie in Aufgabe (1c) entwickelt haben. (Dafür müssen natürlich auch die Datentypen, die sie in (1c) entwickelt haben, in `Ue6A1main.cpp` bekannt sein.)

   Compilieren Sie das Ergebnis! Treten Fehler auf? Falls ja: Erklären Sie diese! Falls nein: Kommentieren Sie alle Aufrufe an `f` außerhalb von `Ue6A1main.cpp` aus (insbesondere auch die, die Sie in Aufgabe (1c) benutzen) und versuchen Sie noch einmal zu compilieren! Erklären Sie Ihre Beobachtungen!

   Hinweis: Suchen Sie nicht nach realweltlichem Sinn in `f`, darum geht es in dieser Aufgabe nicht. Es geht nur darum, das Zusammenspiel zwischen einem Template-Konstrukt und seinen Realisierungen an einem Beispiel durchzugehen.

2. In dieser Aufgabe geht es um die Entwicklung einer Template-Klasse `FlexArray`, die ähnliche Aufgaben wie ein Array übernehmen kann, aber gegenüber dem Array den Vorteil bietet, dass man immer auf jeden nichtnegativen Index zugreifen kann, ohne dass dadurch ein Speicherzugriffsfehler entsteht.

   Um das Verhalten von Arrays zu spiegeln, stellt `FlexArray` zwei Methoden zur Verfügung, nämlich `get` und `set`, mit denen, unter Angabe eines Index, aus dem Array ausgelesen, bzw. in das Array geschrieben werden kann.

   Die zugrundeliegende Datenhaltung geschieht dabei über ein richtiges Array, das wir im Folgenden `data` nennen.

   Beispiel 1: Angenommen wir hätten ein normales `int` Array `a` der Größe 5. Dann kann über `a[3]` zum Beispiel das 3. Element des Arrays ausgelesen werden. Diese Funktionalität wird in `FlexArray` durch einen Aufruf der Form `get(3)` gespiegelt. Dazu liest die Methode `get` einfach das 3. Element von `data` aus und gibt es zurück.

Beispiel 2: Angenommen `a` ist wieder ein `int` Array der Größe 5. Ein Zugriff der Form `a[7]` greift dann über das Array hinaus und stellt eine Speicherverletzung dar. Im `FlexArray` habe `data` ebenfalls die Größe 5. Ein Zugriff der Form `get(7)` soll aber nicht zu einer Speicherverletzung oder eine Fehlermeldung führen, sondern folgendes Verhalten auslösen:

- Es wird ein neues Array mit mehr als 7 Elementen erzeugt.
- Der Inhalt von `data` wird in das neue Array kopiert.
- Da das neue Array größer ist als `data` enthält es Elemente, die durch den Kopiervorgang des letzten Schrittes nicht gesetzt wurden. Diese Elemente sollen mit einem Defaultwert initialisiert werden.
- Der Speicher für `data` wird freigegeben.
- `data` wird auf das neue Array gesetzt.
- Die Methode `get` gibt nun den Inhalt des neuen `data` an der Stelle 7 zurück.

Die `set`-Methode arbeitet mit einem analogen Mechanismus. Auf diese Weise kann ein Benutzer von `FlexArray` auf jeden beliebigen nichtnegativen Index zugreifen, ohne, dass je eine Speicherverletzung stattfinden kann.

Außerdem soll `FlexArray` als Template-Klasse mit den Template-Parametern `T, sizeFakt` und `initialSize` entwickelt werden. Dabei gibt `T` an, von welchem Typ die in einer entsprechenden Realisierung von `FlexArray` gespeicherten Elemente sind. `initialSize` ist ein Template-Parameter vom Typ `unsigned int` und gibt, an, wie groß das dem `FlexArray` zugrundeliegende richtige Array `data` zum Konstruktionszeitpunkt sein soll.

Schließlich bestimmt `sizeFakt` (ebenfalls vom Typ `unsigned int`), um wie viel `data` vergrößert werden soll, wenn durch `get` oder `set` auf einen Index außerhalb des aktuellen Bereichs von `data` angesprochen wird. Ist z.B. `sizeFakt` $= 2$, so wird ein neues Array gebaut, dass doppelt so groß ist wie `data`.

Das `FlexArray` sollte mindestens drei Attribute besitzen: Das oben besprochene `data`, das ein Array vom Typ `T` ist. Die aktuelle Größe dieses Arrays sollte in einem Attribut `size` gespeichert werden. Schließlich brauchen wir noch ein Attribut vom Typ `T` für den oben angesprochenen Default-Wert.

Aufgaben:

(a) Legen Sie eine Template-Klasse für `FlexArray` mit den oben angesprochenen Template-Parametern und Attributen an. Schreiben Sie einen Konstruktor für diese Klasse, die ein einziges Argument mit Namen `defaultV` vom Typ `T` erhält. Der Konstruktor initialisiert `data` gemäß seiner Template-Parameter, befüllt es mit `defaultV` und merkt sich den Default-Wert sowie die Array-Größe in den dafür vorgesehenen Attributen.
Schreiben Sie auch einen dazu passenden Destruktor!
Legen Sie eine neue `.cpp`-Datei an, die eine `main`-Funktion enthält, in der unterschiedliche Realisierungen von `FlexArrays` initialisiert werden. Compilieren Sie Ihren Code!

(b) Schreiben Sie eine private Methode

```
void enlarge()
```

die das `data`-Array im oben besprochenen Sinn vergrößert.

(c) Schreiben Sie eine öffentliche Methode `get`, die als einziges Argument einen Index vom Typ `unsigned int` erhält. Die Methode greift im oben beschriebenen Sinn auf `data` zu und liefert das entsprechende Element zurück. Sollte der angegebene Index zu groß sein, wird `data` zuvor über `enlarge` vergrößert.

(d) Schreiben Sie eine öffentliche Methode `set`, die zwei Argumente erhält, zum einen einen Index vom Typ `unsigned int` und zum anderen einen Wert vom Typ `T`. Die Methode schreibt den Wert im oben beschriebenen Sinn an die indizierte Stelle in `data`. Sollte der angegebene Index zu groß sein, wird `data` zuvor über `enlarge` vergrößert.
Testen Sie Ihre `get`- und `set`-Methoden und untersuchen Sie ihr Verhalten, wenn auf Elemente mit Indizes zugegriffen wird, die mal im Rahmen der aktuellen `size` liegen und mal nicht!

(e) (Bonusaufgabe)
Schreiben Sie eine öffentliche Methode mit dem Kopf

```
void exchange(unsigned int i, unsigned int j)
```

die die beiden Elemente mit den Indizes `i` und `j` im Array vertauscht. Achten Sie darauf, dass `i` oder `j` (oder beide) auf zunächst nicht zulässige Weise adressieren könnten. In dem Fall muss `data` wieder über `enlarge` vergrößert werden!