technische universität dortmund

fakultät für informatik

Jurij Kuzmic
Horst Schirmeier
Jonas Ellert

Dortmund, den 14. Januar 2021

# Übung zur Vorlesung EidP (WS 2020/21)

# Blatt 8

Block rot

**Es können 4 Punkte erreicht werden.**

**Abgabedatum:** 21. Januar 2021, 23:59 Uhr

## Hinweise

- Bitte beachten Sie aktuelle Hinweise unter:

  `https://ls11-www.cs.tu-dortmund.de/teaching/ep2021uebung/`

- Verwenden Sie **keine zusätzlichen Bibliotheken** zur Lösung der Aufgaben.
- Sie sollten während der Entwicklung Ihrer Programme diese unbedingt regelmäßig – und insbesondere noch einmal vor der Abgabe – **compilieren und ausführen**.
- Auf der Übungswebseite wurde eine **Probeklausur** veröffentlicht.
- Auf der Übungswebseite wurde ein **Zusatzblatt mit 6 Bonuspunkten**
- **Die Präsenzaufgabe entfällt auf diesm Blatt, sodass stattdessen Aufgaben der Probeklausur besprochen werden können**. Bitte teilen Sie Ihrem Übungsgruppenleiter rechtzeitig mit, welche Aufgaben Sie besprechen möchten.

## Aufgaben

### Aufgabe 1: Grundlagen (1 Punkt)

Legen Sie für Ihre Antworten eine Text-Datei `Aufgabe_08_1.txt` an.

**a)** Wofür steht die Abkürzung ADT? Nennen Sie mindestens drei Eigenschaften von ADT. (0.3 Punkte)

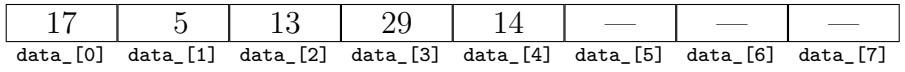
**b)** Welche Gemeinsamkeiten und Unterschiede haben der Kopierkonstruktor und der Zuweisungsoperator? Welche wichtige Funktionalität kann (und häufig auch muss) der Zuweisungsoperator im Gegensatz zum Kopierkonstruktor erfüllen? (0.3 Punkte)

**c)** Erklären Sie die Begriffe der flachen bzw. tiefen Objektkopien. (0.2 Punkte)

**d)** Wie kann man automatisch erzeugte Funktionen einer Klasse erzwingen bzw. verhindern? Geben Sie jeweils einen Beispiel an. (0.2 Punkte)

## Aufgabe 2: ADT Stapel (3 Punkte)

Laden Sie für diese Aufgabe die Dateien `Aufgabe_08_2.cpp` und `simple_stack.hpp` von der Übungswebseite herunter. Aus Kapitel 9 der Vorlesung kennen Sie bereits den ADT *Stapel*. In der Datei `simple_stack.hpp` finden Sie die Deklaration einer weiteren Version des Stapels. Im Arbeitsspeicher verwendet diese Version das gleiche Layout wie die Version aus der Vorlesung: Die auf dem Stapel liegenden Elemente werden in einem Array `data_` der Größe `capacity_` gespeichert. Die Variable `size_` gibt an, wie viele Elemente aktuell vorhanden sind. Das unterste Element (also das Element, das zuerst hinzugefügt wurde) steht an Position `data_[0]`. Das oberste Element (also das Element, das zuletzt hinzugefügt wurde) steht an Position `data_[size_ - 1]`. Nach dem Hinzufügen der Elemente $17, 5, 13, 29, 14$ würde ein Stack mit `capacity_ = 8` zum Beispiel wie folgt aussehen:



(`size_ = 5`)

**a)** Ergänzen Sie in der Datei `simple_stack.hpp` die Implementierung der Konstruktoren, des Destruktors, und des Zuweisungsoperators. Der Konstruktor ohne Parameter soll einen leeren Stapel mit `capacity_ = 1` erstellen. Der Kopierkonstruktor und der Zuweisungsoperator sollen eine *tiefe* Kopie des übergebenen Stapels erstellen. Achten Sie darauf, an den richtigen Stellen mit `new[]` und `delete[]` Speicher dynamisch zu allokieren und wieder freizugeben. Sie dürfen davon ausgehen, dass der Aufruf von `new[]` und `delete[]` immer fehlerfrei Funktioniert (keine Ausnahmebehandlung nötig). (1.2 Punkte)

**b)** Ergänzen Sie in der Datei `simple_stack.hpp` die Implementierung der Funktionen `push` und `pop`.

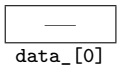
Die Funktion `push` legt das übergebene Element auf den Stapel (i.e. schreibt das Element an Position `data_[size]`) und erhöht anschließend `size_` um 1. Überprüfen Sie zuvor, ob für das neue Element Platz vorhanden ist, oder ob die Größe des Stapels bereits die maximale Kapazität erreicht hat. Wenn kein Platz vorhanden ist, dann sollten Sie die Kapazität verdoppeln. Dazu müssen Sie ein neues Array doppelter Größe anlegen, und die vorhandenen Elemente aus dem alten Array in das neue Array kopieren. Denken Sie daran, nicht länger benötigte Speicherbereiche wieder freizugeben! Ein Beispiel zu `push` finden Sie auf Seite 3.

Die Funktion `pop` entfernt das oberste Element vom Stapel. Wenn der Stapel beim Aufruf von `pop` leer ist, dann nutzen Sie die Funktion `error`, um eine entsprechende Fehlermeldung auszugeben. Sie sollten außerdem verhindern, dass der Stapel *zu wenige Elemente relativ zu seiner Kapazität* enthält. Wenn beim Aufruf von `pop` nur ein Viertel (oder weniger) des allokierten Speicherplatzes verwendet wird, dann sollten Sie die Kapazität halbieren. Da Sie also sowohl für `push` als auch für `pop` möglicherweise die Kapazität anpassen müssen, bietet es sich an, die Kapazitätsänderung in der privaten Hilfsfunktion `set_capacity` zu implementieren.
(1.2 Punkte)

**c)** Implementieren Sie die Funktionen `size` (gibt die aktuelle Anzahl der Elemente zurück), `top` und `bottom`. Die Funktion `top()` ohne Parameter gibt das oberste Element des Stapels zurück. Die Funktion `top(x)` gibt hingegen das $(x + 1)$-te Element von oben zurück (sodass die Aufrufe `top()` und `top(0)` identisch sind). Die Funktion `bottom()` ohne Parameter gibt das unterste Element des Stapels zurück. Die Funktion `bottom(x)` gibt hingegen das $(x + 1)$-te Element von unten zurück (sodass die Aufrufe `bottom()` und `bottom(0)` identisch sind). Ein Beispiel zu `top` und `bottom` finden Sie auf Seite 3. Bei ungültigen Aufrufen sollten Sie mithilfe der `error`-Funktion passende Fehlermeldungen ausgeben. (0.6 Punkte)

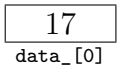
**Beispiel: (zu Aufgabe 2b)**
Sie erstellen einen neuen Stack.

`size_` $= 0$, `capacity_` $= 1$



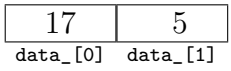

Nun fügen Sie ein Element ein: `push(17)`

`size_` $= 1$, `capacity_` $= 1$



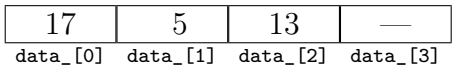

Sie fügen ein weiteres Element ein: `push(5)`. Da Sie bereits die maximale Kapazität erreicht haben, müssen Sie zunächst die Kapazität verdoppeln, und erst dann 5 einfügen.

`size_` $= 2$, `capacity_` $= 2$



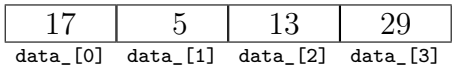

Sie fügen ein weiteres Element ein: `push(13)`. Erneut haben Sie die maximale Kapazität erreicht, und müssen diese daher verdoppeln.

`size_` $= 3$, `capacity_` $= 4$



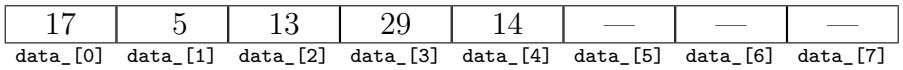

Sie fügen ein weiteres Element ein: `push(29)`.

`size_` $= 4$, `capacity_` $= 4$




Sie fügen ein weiteres Element ein: `push(14)`. Sie haben die maximale Kapazität erreicht, und müssen diese daher erneut verdoppeln.

`size_` $= 5$, `capacity_` $= 8$




Vor der nächsten Verdopplung der Kapazität könnten Sie noch drei weitere Elemente einfügen.

**Beispiel: (zu Aufgabe 2c)**
Nach dem letzten Schritt des vorherigen Beispiels gilt:

```
top() == 14
top(1) == 29
top(4) == 17
bottom() == 17
bottom(2) == 13
bottom(3) == 29
```