Praktikum 5: Klassen und dynamisch erzeugter Speicher

1 Lernziele

- Erstellen einer Klasse mit Konstruktoren und Destruktor
- Verwenden von dynamisch erzeugtem Speicher
- Verständnis der Funktionsweise von Datenstrukturen aus der Standardbibliothek

2 Aufgabenstellung

Erstellen Sie eine Klasse `DynArray`, die ein automatisch wachsendes Array ähnlich der `vector`-Klasse bereit stellt. Dabei soll nur ein kleiner Teil der Funktionalität der `vector`-Klasse implementiert werden.

Die Klasse verwaltet dazu ein dynamisch erzeugtes Array, welches bei Bedarf durch ein größeres ersetzt wird. Damit nicht ständig das Array um ein Element vergrößert werden muss, ist es notwendig, dass das Array stets ein paar Plätze „in Reserve" hat. Daher gibt es zwei voneinander verschiedene **Größen**, mit denen ein `DynArray`-Objekt arbeitet:

capacity ist die physikalische oder tatsächliche Größe des Arrays, und damit gleichzeitig die maximale Anzahl an Elementen die ein `DynArray`-Objekt aufnehmen kann ohne das Array zu vergrößern.

size ist die Anzahl an tatsächlich in einem `DynArray`-Objekt vorhandenen Elemente

Die beiden Bezeichnungen sind analog zu den Bezeichnungen in der `vector`-Klasse gewählt. Dort existieren diese beiden Größen ebenfalls, und sind über die Methoden `size()` und `capacity()` abrufbar.

Die Klassendeklaration und die Implementierung sind in eine `.h` und eine `.cpp` Datei aufzutrennen. Implementierung von Methoden im Headerfile wird nicht akzeptiert.

2.1 Klassendeklaration

Verwenden Sie folgende Klassendeklaration:

```
12 class DynArray
13 {
14 public:
15   DynArray(); //Standardkonstruktor
16   DynArray(int newCapacity); //Weiterer Konstruktor
17   ~DynArray(); //Destruktor, muss implementiert werden
18
19   Lied& at(int index);
20
21   void push_back(Lied elem);
22   void pop_back();
23   void erase(int index);
24
25   int size();
26   int capacity();
27
28   void print();
29
30 private:
31   void resize(int newCapacity);
32   DynArray(const DynArray& other); //Kopierkonstruktor soll explizit *nicht* aufgerufen werden
33   Lied* m_data;
34   int m_size;
35   int m_capacity;
36 };
```

Sie finden die oben genannte Deklaration in der Datei `DynArray.h` in Moodle.

2.2 Konstruktoren & Destruktoren

Dies bezeichnet „spezielle“ Methoden, die automatisch aufgerufen werden wenn ein Objekt angelegt bzw. zerstört wird. Innerhalb dieser Funktionen können Attribute der Klasse initialisiert werden.
Wenn eine Klasse mehrere Konstruktoren anbietet, so können verschiedene Initialisierungen vorgenommen werden:

```
1 DynArr a; //verwendet den Standardkonstruktor
2 DynArr b(10); //verwendet den zweiten Konstruktor
```

Je nachdem, welche Initialisierung beim Anlegen des Objektes verwendet wurde, wird der entsprechende Konstruktor automatisch aufgerufen.

Die Konstruktoren sollen folgende Aufgaben erfüllen:

`DynArray(int newCapacity)` dient dazu, ein `DynArray`-Objekt mit einer bestimmten Kapazität anzulegen:

- Erzeugen eines neuen Arrays mit der Größe `newCapacity`
- Die Attribute `capacity` und `size` sollen entsprechend gesetzt werden
  - `capacity` ist immer gleich der Größe des Arrays, also die maximal mögliche Anzahl an Elementen
  - `size` ist immer die Anzahl an tatsächlichen Elementen, die das `DynArray`-Objekt enthält
- Ein auf diese Art angelegtes `DynArray`-Objekt soll **leer** sein im Sinne von „es enthält keine Elemente“

`DynArray()` erfüllt fast die gleichen Aufgaben wie der eben genannte Konstruktor, allerdings wird im Quellcode eine Kapazität fest vorgegeben. Diese können Sie beliebig wählen.

Hinweis: Sie können als Alternative zur vorgegebenen Klassendeklaration auch **einen** Konstruktor mit einem geeigneten Default Value verwenden.

2.2.1 Destruktor

Der Destruktor ist eine spezielle Methode, die automatisch vom Programm aufgerufen wird, wenn ein Objekt der Klasse zerstört wird. Hier **müssen** Sie die durch die Klasse dynamisch erzeugten Ressourcen wieder freigeben.

2.3 Methodenübersicht

2.3.1 `double& at(int index)`

Geben Sie eine Referenz auf das Element mit dem angegebenen Index zurück. Sie müssen nicht überprüfen, ob der Index außerhalb des gültigen Bereiches liegt.
Frage: Warum muss der Rückgabedatentyp `double&` sein, warum reicht `double` nicht aus?

2.3.2 `int size()`

Gibt eine Kopie des Attributs `size` zurück, also die Anzahl der Elemente, die tatsächlich in einem `DynArray`-Objekt gespeichert sind.

2.3.3 `int capacity()`

Gibt eine Kopie des Attributs `capacity` zurück, also die tatsächliche Größe des Arrays. Dies ist gleichzeitig die maximal mögliche Anzahl an Elementen, die das `DynArray`-Objekt speichern kann, ohne das das Array vergrößert werden muss.

2.3.4 `void print()`

Soll alle tatsächlich gespeicherten Elemente auf der Konsole ausgeben.

2.3.5 `void resize(int newCapacity)`

Ein Aufruf von `resize` vergrößert das Array eines `DynArray`-Objekts. Diese Funktion wird nur intern verwendet und ist daher `private`. Um ein bestehendes Array zu vergrößern müssen Sie folgende Schritte durchführen:

1. Erzeugen eines neuen dynamischen Arrays der Größe `newCapacity`
2. Kopieren der Daten aus dem bestehenden Array in das neu erzeugte
3. Löschen des bestehenden Arrays
4. Den `m_data`-Zeiger auf das neu erzeugte Array setzen

Hinweis: Überlegen Sie, ob ein Aufruf dieser Funktion Auswirkungen auf Attribute der Klasse hat.

2.3.6 `void push_back(double elem)`

Diese Funktion fügt ein neues Element zum Array hinzu. Wenn der restliche Platz im Array nicht ausreicht, also die Kapazität erreicht ist, muss das Array vergrößert werden. Hierzu sollten Sie die gerade beschriebene Funktion `resize` verwenden, die Ihnen genau dieses Problem löst.

Hinweis: Wenn Sie das Array vergrößern, sollten Sie aus Effizienzgründen die Kapazität nicht nur um eins erhöhen. Dies würde dazu führen, dass andauernd die Arraygröße angepasst wird. Erhöhen Sie statt dessen um einen größeren Wert, oder verdoppeln Sie die aktuelle Kapazität.

2.3.7 `pop_back()`

Mittels dieser Funktion soll das letzte Element des dynamischen Arrays als gelöscht angesehen werden. Überlegen Sie, auch im Sinne von „Freigeben von Speicher", welche Aktionen dazu notwendig sind.
Nach dieser Aktion ist das dynamische Array um ein Element kleiner.

2.3.8 `erase(int index)`

Diese Funktion soll das Element am angegebenen `index` aus dem dynamischen Array entfernen. Hierzu ist es auch notwendig, alle nachfolgenden Elemente des Arrays entsprechend zu verschieben.
Nach dieser Aktion ist das dynamische Array um ein Element kleiner.

2.4 Fortlaufende Aufgabe

Ersetzen Sie den `vector` in Ihrer Musikdatenbank durch Ihre `DynArray`-Klasse und testen Sie, ob alle Funktionalitäten weiterhin korrekt sind.