

Programmieren C++: Function Pointer

Klaus Kusche

*Einfache Array-Klasse mit **qsort** und **apply***

Wir wollen

- eine Methode schreiben, die eine beliebige, als Parameter übergebene Funktion auf alle Elemente eines Arrays anwendet
- und eine weitere Methode, die das Array unter Verwendung der vordefinierten Funktion **qsort** sortiert.

Als Vorarbeit schreiben wir eine ganz einfache Klasse für variabel große **double**-Arrays:

- Die Klasse hat drei Member:
 - Einen Pointer auf **double**: Er zeigt auf das im Konstruktor dynamisch mit **new** angelegte Array (wo wird es wieder freigegeben?).
 - Einen **unsigned int** für die Größe des Arrays (sie wird beim Aufruf des Konstruktors als Parameter angegeben).
 - Noch einen **unsigned int** für die Anzahl der tatsächlich belegten Elemente im Array (bei einem neu angelegten Array **0**).
- Die Klasse hat einen Konstruktor (er wird mit der gewünschten Anzahl der Elemente aufgerufen und legt ein Array dieser Größe mit **new** an, aber lässt die einzelnen Elemente uninitialized) und einen Destruktor (was macht der?).
- Weiters hat die Klasse eine Methode **append**: Sie wird mit einem **double**-Wert als Argument aufgerufen, der hinten an die schon belegten Elemente im Array angehängt werden soll (die Anzahl der belegten Elemente wächst daher um eins). Sie liefert ein **bool**-Ergebnis: **true** bei Erfolg und **false**, wenn das Array schon voll war.
- Dann schreiben wir für die Objekte der Klasse noch einen Ausgabe-Operator << : Er soll alle belegten Elemente des Arrays in einer Zeile ausgeben.

Dazu schreiben wir zum Testen ein Hauptprogramm. Es soll

- ein Objekt unserer Array-Klasse mit **argc-1** Elementen anlegen,
- alle Werte auf der Befehlszeile der Reihe nach in **double**'s verwandeln und mit **append** an unser Array-Objekt anhängen,
- und schließlich das Array mit << ausgeben.

Klappt das, erweitern wir unsere **Array**-Klasse um eine Methode "apply", die eine beliebige Funktion mit einem double-Parameter und Returntyp double der Reihe nach auf alle Elemente des Arrays anwendet:

Die Funktion wird in einer Schleife für jedes Array-Element einzeln aufgerufen, und ihr Returnwert wird gleich wieder in demselben Element gespeichert.

Der Pointer auf die anzuwendende Funktion wird der **apply**-Methode als einzigster Parameter übergeben, die Methode hat keinen Returnwert.

Zum Test unserer Methode erweitern wir unser Hauptprogramm um eine Schleife:

- Zuerst wird der aktuelle Inhalt des Arrays ausgegeben.
- Dann fragen wir den Benutzer nach einem Buchstaben.
- Ist der eingelesene Buchstabe 'x', so endet das Programm.
- Ist der Buchstabe 'w', 'q', 'l' oder 'e', so berechnen wir mit unserer **apply**-Methode von allen Elementen des Arrays entweder die Wurzel (vordefinierte Funktion **sqrt**), oder das Quadrat (selbstgeschriebene Funktion **square**, **double**-Argument und **double**-Returnwert), oder den Logarithmus (vordefinierte Funktion **log**) oder die e-Funktion (vordefinierte Funktion **exp**). (**cmath** inkludieren!)

Klappt das auch, erweitern wir unsere Klasse nochmals, und zwar um eine Methode "**sort**" (ohne Argumente und ohne Returnwert), die das Array aufsteigend sortieren soll.

Intern soll diese Methode die vordefinierte C-Funktion qsort (aus **stdlib**) zum Sortieren der Elemente verwenden, keinen selbstgeschriebenen Sortieralgorithmus.

qsort wird mit 4 Argumenten aufgerufen:

- Dem zu sortierenden Array.
- Der Anzahl der Elemente im Array.
- Der Größe eines Elementes (in unserem Fall: Eines **double**'s) in Bytes (mit welchem C-Sprachkonstrukt bekommst du diese Größe?).
- Einem Pointer auf eine Funktion, die von **qsort** intern immer wieder aufgerufen wird, um zwei beliebige Array-Elemente zu vergleichen.

Diese Vergleichs-Funktion müssen wir selbst schreiben (mach dich notfalls am Internet in der Doku von **qsort** schlau, wie diese Funktion auszusehen hat):

- Sie bekommt zwei "const void"-Pointer auf die beiden zu vergleichenden Array-Elemente übergeben.

Wir müssen diese beiden Pointer in "**const double**"-Pointer verwandeln (wie macht man das?) und uns die beiden **double**-Werte holen, auf die diese Pointer zeigen.

- Die Vergleichsfunktion muss so wie **strcmp** einen int-Returnwert liefern: **0** wenn beide Werte gleich sind, eine negative Zahl (**-1**) wenn der erste Wert kleiner als der zweite ist, und eine positive (**1**), wenn er größer ist.

Zum Testen erweitern wir nur die Schleife unseres Hauptprogrammes um einen Fall: Ist der eingegebene Buchstabe 's', wird kein **apply** aufgerufen, sondern das Array mit unserer **sort**-Methode sortiert.