GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
AUFSTIEG DURCH BILDUNG >>
OFFENE HOCHSCHULEN
ESF Europäischer Sozialfonds für Deutschland
EUROPÄISCHE UNION

Programmieren mit C++

Konrad Doll

Technische Hochschule Aschaffenburg
University of Applied Sciences Aschaffenburg
Version 1.2

Einleitung

In der Kurseinheit „Programmieren mit C++“ sollen die Grundlagen der objektorientierten Programmierung mit der Programmiersprache C++ erlernt werden. Dieser Kurs setzt auf den Kenntnissen der Kurseinheit „Programmieren mit C“ auf und ergänzt diese insbesondere um die objektorientierte Konzepte: Abstraktion, Kapselung, Hierarchie und Polymorphismus. Der Kurs basiert auf einem Blended-Learning-Konzept. Zu den Unterrichtsmaterialien gehört ein E-Learning-Angebot, ein Lehrbrief und zwei Präsenzphasen. Im Rahmen des E-Learning-Angebots werden die einzelnen Konzepte anhand von Beispielen detailliert besprochen. Zusätzlich soll die Aufmerksamkeit und die Lernbereitschaft der Bearbeiter durch regelmäßige Fragen zum Thema gesteigert werden. Die dabei genannten Konzepte werden in einem Lehrbrief zusammenfassend und möglichst knapp dargestellt. Deswegen und weil das E-Learning-Angebot bereits umfangreiche Fragen zum Verständnis enthält, sind im Lehrbrief keine Fragen enthalten. In den Präsenzphasen werden Aufgaben besprochen und es kann auf individuelle Fragen eingegangen werden. Anhand von Übungsaufgaben werden die gelernten Kenntnisse und Fähigkeiten vertieft. Die Aufgabenstellungen werden in schriftlicher Form kommuniziert. Erläuterungen finden über die E-Learning-Plattform statt.

Der vorliegende Kurs ist folgendermaßen gegliedert: Zunächst erfolgt in Kapitel 1 eine Einführung in C++. Mit Hilfe der Programmiersprache C werden die oben genannten objektorientierten Konzepte eingeführt. Darüber hinaus werden an einfachen Beispielen erste Schritte in C++ unternommen. Kapitel 2 beschreibt die Unterschiede zwischen C und C++, wenn man objektorientierte Aspekte außer Acht lässt. Kapitel 3 geht auf die Abstraktion mit Elementfunktionen ein. Die Kapselung steht im Kapitel 4 im Fokus. Auf das Konzept der Hierarchie insbesondere auf die Vererbung wird in Kapitel 5 eingegangen. Kapitel 6 und 7 widmen sich dem Erzeugen und Löschen von Objekten mit den Konstruktoren und Destruktoren sowie mit new und delete. Referenzen werden in Kapitel 8 besprochen. Auf spezielle Elementfunktionen wie Kopierkonstruktor, Zuweisungsoperator oder Ausgabeoperator wird in Kapitel 9 eingegangen. Templates stehen in Kapitel 10 im Mittelpunkt des Interesses, bevor in Kapitel 11 der Polymorphismus besprochen wird. Aspekte der Ein- und Ausgabe werden in Kapitel 12 vorgestellt. Im abschließenden Kapitel 13 erfolgt eine Zusammenfassung anhand eines Beispiels.

In folgenden Büchern können detaillierte Informationen gefunden werden, die über das in dieser Kurseinheit Vermittelte weit hinausgehen: [SB95, Bre11, Bra10a, Bra10b, Dan98, Eck00, Wol09, Str11, RRZ11, Lui10].

Lernziele

Ziel dieser Kurseinheit ist es, Kenntnisse, Fähigkeiten und Methoden beim objektorientierten Programmieren mit C++ zu vermitteln.

Nach dem Studium dieser Kurseinheit sollten Sie:

- objektorientierte Konzepte (Abstraktion, Kapselung, Hierarchie und Polymorphismus) verstehen und den Unterschied zur Programmiersprache C begreifen,
- die Fähigkeit haben, eine informationstechnischen Aufgabenstellung mit der objektorientierten Programmiersprache C++ zu lösen und
- Funktionen von Betriebssystemen und Entwicklungsumgebungen nutzen können.

Inhaltsverzeichnis

Kapitel 3

Abstraktion mit Elementfunktionen

In diesem Kapitel wollen wir detailliert das Konzept der Abstraktion in C++ besprechen und dazu Elementfunktionen benutzen. Zu diesem Zweck implementieren wir für einen abstrakten Datentyp, der aus einem Array aus Integer Zahlen besteht, eine Klasse in C++. Dieses Beispiel ist [SB95] entnommen.

3.1 Definition einer Klasse

Zunächst erzeugen wir in einer Header-Datei „`array.h`“ eine Definition der Klasse `IntArray` für den abstrakten Datentyp.

Beispiel:

```
1  #ifndef ARRAY_H
2  #define ARRAY_H
3  #include <cstddef>
4
5  class IntArray
6  {
7     public:
8        void init();
9        void cleanup();
10       void setSize(size_t value);
11       size_t getSize();
12       void setElem(size_t index, int value);
13       int getElem(size_t index);
14    private:
15       int *elems;
16       size_t numElems;
17 };
18
```

```
19 #endif
```

Listing 3.1: `Definition Klasse IntArray`

Erläuterung:

- Zeilen 1, 2, 19: Mit den Präprozessor-Anweisungen
  `#ifndef ARRAY_H`
  `#define ARRAY_H`
  `...`
  `#endif`
  wird sichergestellt, dass die Datei nur einmal eingebunden wird.

- Zeile 3: Die Datei „`cstdef`“ wird eingebunden, damit `size_t` benutzt werden kann.

- Zeilen 5, 6, 17: Die Klasse `IntArray` wird mit dem Schlüsselwort `class` definiert. In der Klasse werden Funktionen und Daten vereinbart. Die Funktionen in einer Klasse heißen Elementfunktionen oder Methoden. Die Daten in einer Klasse heißen Elementdaten oder Attribute.

- Zeile 7: Das Schlüsselwort `public` gibt an, dass die folgenden Elementdaten oder Elementfunktionen öffentlich sind, d. h., man kann auf diese Elemente von überall im Programm aus zugreifen. Häufig sind die Elementfunktionen öffentlich (wie auch in diesem Beispiel). Die Menge aller Elementfunktionen, die öffentlich sind, bilden die Schnittstelle der Klasse.

- Zeilen 8 bis 13 : Hier werden die Elementfunktionen deklariert. Es gelten die üblichen Regeln für die Deklaration der Funktionen. Es handelt sich in diesem Beispiel um Funktionen zum Initialisieren (`init()`), zum Löschen (`cleanup()`), zum Setzen der Größe des Arrays (`setSize(size_t value)`), zum Auslesen der Größe des Arrays (`getSize()`), zum Setzen eines Elements des Arrays (`setElem(size_t index, int value)`) und zum Auslesen des Elements eines Arrays (`getElem(size_t index)`).

- Zeile 14: Das Schlüsselwort `private` gibt an, dass die folgenden Elementdaten oder Elementfunktionen privat sind, d. h., man kann auf diese Elemente nur von Elementfunktionen dieser Klasse aus zugreifen. Häufig sind die Elementdaten privat (wie auch in diesem Beispiel).

- Zeilen 15, 16: An dieser Stelle werden die Elementdaten vereinbart. In unserem Fall bestehen die Elementdaten aus einem Zeiger auf die Elemente des Arrays (`int *elems`) und aus einer Variablen, die die Anzahl der Elemente des Arrays angibt (`size_t numElems`).

3.2 Aufruf der Elementfunktionen

Nun gehen wir darauf ein, wie die Elementfunktionen der Schnittstelle der Klasse `IntArray` aufgerufen werden.

Beispiel: Die folgenden Anweisungen stehen z. B. in der Datei „`main.cpp`“.

```
1  #include "array.h"
2
3  int main()
4  {
5      IntArray powersOf2;
6      powersOf2.init();
7      // ...
8      powersOf2.setSize(8);
9      powersOf2.setElem(0, 1);
10     powersOf2.setElem(1, 2 * powersOf2.getElem(0));
11     powersOf2.setElem(2, 2 * powersOf2.getElem(1));
12     // ...
13     powersOf2.cleanup();
14 }
```

Listing 3.2: `Aufruf der Elementfunktionen`

Erläuterung:

- Zeile 1: Zunächst wird die Datei „`array.h`“ eingebunden, damit die Klasse `IntArray` bekannt ist.
- Zeile 5: Nun wird ein Objekt der Klasse `IntArray` mit dem Namen `powersOf2` definiert.
- Zeile 6: Die Elementfunktionen einer Klasse werden mit Hilfe des Objektnamens `powersOf2` (damit bekannt ist, mit welchem Objekt gearbeitet werden soll) und dem Punkt-Operator aufgerufen. Für die Elementfunktionen selbst wird der Name verwendet, der in der Klasse deklariert wurde. Dies geschieht in Anlehnung an den Zugriff auf Elemente einer Struktur in C, der auch mit dem Punkt-Operator erfolgt. In dem Beispiel wird die Elementfunktion `init()` auf dem Objekt `powersOf2` der Klasse `IntArray` aufgerufen, so dass dieses Objekt initialisiert wird.
- Zeilen 8 bis 11: Für das Objekt `powersOf2` wird die Größe festgelegt. Anschließend werden die Potenzen von 2 in das Array geschrieben.
- Zeile 12: Das Objekt `powersOf2` wird mit der Elementfunktion `cleanup()` „aufgeräumt“.

3.3 Implementierung der Elementfunktionen

Nachdem wir die Klasse `IntArray` definiert haben und wissen, wie Elementfunktionen einer Klasse aufgerufen werden, beschäftigen wir uns in diesem Abschnitt mit der Implementierung der Elementfunktionen.

Dazu ist es notwendig, dass wir in einer Elementfunktion auf die Elementdaten des Objekts zugreifen, auf dem die Elementfunktion aufgerufen wurde. Damit dies möglich ist, stellt die Programmiersprache C++ mit dem Schlüsselwort „`this`“ einen Zeiger auf das Objekt zur Verfügung, auf dem die Elementfunktion aufgerufen wurde. Damit kann nun auf die entsprechenden Elementdaten zugegriffen werden.

Beispiel: Die Implementierung der Elementfunktionen erfolgt in der Datei „`array.cpp`“.

```
1  #include <iostream>
2  using namespace std;
3  #include "array.h"
4
5  // exit in case of error
6  void error(char *s)
7  {
8      cout << s;
9      exit(1);
10 }
11
12 // definition of interface functions for IntArray:
13 void IntArray::init()
14 {
15     this->numElems = 0;
16     this->elems = 0;
17 }
18
19 void IntArray::cleanup()
20 {
21     free(this->elems);
22 }
23
24 void IntArray::setSize(size_t value)
25 {
26     if (this->elems != 0) free(this->elems);
27     this->numElems = value;
28     this->elems = (int *) malloc (value * sizeof(int));
29 }
30
31 size_t IntArray::getSize()
32 {
33     return this->numElems;
34 }
```

```
35
36 void IntArray::setElem(size_t index,
37                         int value)
38 {
39    if (index >= this->numElems) error("bad index");
40    this->elems[index] = value;
41 }
42
43 int IntArray::getElem(size_t index)
44 {
45    if (index >= this->numElems) error("bad index");
46    return this->elems[index];
47 }
```

Listing 3.3: Implementierung der Elementfunktionen

Erläuterung:

- Zeile 1 bis 3: Es werden die notwendigen Header-Dateien eingebunden.
- Zeile 5 bis 10: Eine globale Funktion `void error(char *s)` zur Fehlerbehandlung wird definiert.
- Zeile 12 bis 17: Die Elementfunktion `init()` der Klasse `IntArray` wird definiert.
- Zeile 13: In dieser Zeile wird der Funktionskopf angegeben. Die Zugehörigkeit der Elementfunktion `init()` zu der Klasse `IntArray` wird mit dem Scope-Operator („`::`“) angegeben.
- Zeile 12, 13: Mit dem `this`-Zeiger kann während der Initialisierung die Anzahl der Elemente auf Null (`this->numElems = 0`) und der Zeiger auf die Elemente im Array auf den Nullzeiger (`this->elems = 0`) gesetzt werden.
- Zeilen 19 bis 22: In der Elementfunktion `cleanup()` wird der Speicherplatz, der für die Elemente des Arrays allokiert wurde, freigegeben.
- Zeilen 24 bis 29: In der Elementfunktion `setSize(size_t value)` wird die Größe des Arrays gesetzt. Dazu wird in Zeile 26 geprüft, ob schon Speicherplatz allokiert wurde. Wenn dem so ist, dann wird der zuvor allokierte Speicherplatz freigegeben. Anschließend wird in Zeile 27 das Elementdatum `numElems` mit dem entsprechenden Wert belegt und es wird in Zeile 28 der notwendige Speicherplatz mit `malloc()` allokiert. Die Inhalte werden nicht kopiert.
- Zeilen 31 bis 34: Die Anzahl der Elemente in dem Array wird in der Elementfunktion `getSize()` zurückgegeben.

- Zeilen 36 bis 41: Mit Hilfe der Elementfunktion `setElem(size_t index, int value)` wird das Element mit dem Index `index` mit dem Wert `value` belegt (Zeile 40). Davor (Zeile 39) wird noch geprüft, ob der `index` im erlaubten Bereich liegt. Wenn dies nicht der Fall ist, wird die Fehlerbehandlungsfunktion `error()` aufgerufen.
- Zeilen 43 bis 47: In der Elementfunktion `getElem(size_t index)` wird zunächst auch der `index` überprüft (Zeile 45) und anschließend wird der Wert des Elements mit dem Index `index` zurückgegeben.

Hinweis zum `this`-Zeiger: Der `this`-Zeiger kann weggelassen werden, wenn das Objekt, auf dem die Elementfunktion aufgerufen wird, eindeutig identifiziert ist. Dies ist in den meisten Situationen der Fall. Sollte dem nicht so sein, wird in diesem Lehrbrief explizit darauf hingewiesen.

Hinweise zum Scope-Operator „`::`“:

- Ohne Angabe eines Gültigkeitsbereichs bei einer Elementfunktion wird eine globale Funktion definiert.
- Der Scope-Operator „`::`“ ohne Präfix bezeichnet den globalen Gültigkeitsbereich. (Beispiel: `::error(char *s)`)
- Der Gültigkeitsbereich muss nicht angegeben werden, wenn der Gültigkeitsbereich aus dem Kontext hervorgeht.
- Die Anweisung „`using namespace std`“ besagt, dass der Namensraum `std` benutzt wird. Damit können alle Namen aus dem Gültigkeitsbereich `std` (wie `cout`, `cin`, etc.) direkt verwendet werden. Andernfalls müssten qualifizierte Namen (wie `std::cout`, `std::cin`, etc.) benutzt werden.

Hinweis zu den Elementfunktionen:

- Elementfunktionen können wie globale Funktionen überladen werden.

3.4 Zusammenfassung

- Abstraktion kann in C++ elegant mit Klassen erreicht werden. Klassen ermöglichen eine Trennung zwischen Schnittstelle und Implementierung. Der Benutzer verwendet dabei nur die Schnittstelle.
- Der `this`-Zeiger referenziert das Objekt, auf dem die Elementfunktion aufgerufen wurde.
- Der Scope-Operator „`::`“ gibt den Gültigkeitsbereich eines Namens an.