Richard Kaiser
www.rkaiser.de

Objektorientierte Programmierung in C++11/14

Inhalt

1 Die Entwicklungsumgebung 1

1.1 Ein erstes C++-Projekt 1
1.1.1 Ein Projekt für ein Standard-C++-Programm anlegen 1
1.1.2 Ein- und Ausgaben über die Konsole 2
1.1.3 cin/cout vs. printf 4
1.1.4 Den Quelltext auf Header-Dateien aufteilen 5
1.1.5 Ein Projekt für die Lösung der Übungsaufgaben 6
1.1.6 Der Start des Compilers von der Kommandozeile Θ 7
1.2 Der Quelltexteditor 7
1.2.1 Tastenkombinationen 7
1.2.2 Intellisense 8
1.2.3 Die Formatierung des Quelltexts 9
1.2.4 Definitionen einsehen 9
1.2.5 Symbole suchen 10
1.2.6 Namen umbenennen 10
1.3 Die Online-Hilfe (MSDN Dokumentation) 11
1.3.1 Hilfe mit *F1* in Visual Studio 11
1.3.2 Die MSDN-Dokumentation im Internet 12
1.4 Projekte und der Projektmappen-Explorer 13
1.4.1 Projekte, Projektdateien und Projektoptionen 13
1.4.2 Projektmappen und der Projektmappen-Explorer 14
1.5 Weiterführende Möglichkeiten Θ 15
1.5.1 Navigieren 15
1.5.2 Code-Ausschnitte 17
1.5.3 Aufgabenliste 17
1.5.4 Der Objektkatalog und die Klassenansicht Θ 17
1.5.5 Die Fenster von Visual Studio anordnen Θ 17
1.5.6 Einstellungen für den Editor Θ 18

2 Elementare Datentypen und Anweisungen 19

2.1 Ganzzahldatentypen 19
2.1.1 Ganzzahlliterale und ihr Datentyp 21
2.1.2 Typ-Inferenz: Implizite Typzuweisungen mit *auto* 23
2.1.3 Der Datentyp *bool* 23
2.2 Kontrollstrukturen und Funktionen 26
2.2.1 Werte- und Referenzparameter 26
2.2.2 Die Verwendung von Bibliotheken und Namensbereichen 26
2.2.3 Default-Argumente 28
2.3 Gleitkommadatentypen 29
2.3.1 Der Datentyp von Gleitkommaliteralen 30
2.3.2 Mathematische Funktionen 31
2.4 Konstanten 32
2.4.1 Laufzeitkonstanten mit *const* 32
2.4.2 Compilezeit-Konstanten mit *constexpr* 33
2.4.3 *constexpr* Funktionen Θ 34
2.5 Exception-Handling Grundlagen: *try*, *catch* und *throw* 35

2.6 Namensbereiche – Grundlagen 38

3 Die Stringklassen: *string*, *wstring* usw. 41

3.1 Die Definition von Variablen eines Klassentyps 42
3.2 Einige Elementfunktionen der Klasse *string* 43
3.3 Raw-String-Literale (Rohzeichenfolgen) 49
3.4 Konversionen zwischen *string*/*wstring* und elementaren Datentypen 50

4 Einfache selbstdefinierte Datentypen 53

4.1 Mit *struct* definierte Klassen 53
4.2 Aufzählungstypen 56
4.2.1 Schwach typisierte Aufzählungstypen (C/C++03) 57
4.2.2 *enum* Konstanten und Konversionen Θ 58
4.2.3 Stark typisierte Aufzählungstypen (C++11) 59

5 Zeiger, Strings und dynamisch erzeugte Variablen 61

5.1 Die Definition von Zeigervariablen 62
5.2 Der Adressoperator, Zuweisungen und generische Zeiger 64
5.3 Dynamisch erzeugte Variablen 66
5.3.1 *new* und *delete* 66
5.3.2 Der Unterschied zu „gewöhnlichen“ Variablen 68
5.3.3 Memory Leaks in Visual C++ finden Θ 71
5.4 Dynamische erzeugte eindimensionale Arrays 72
5.5 Funktionszeiger und Datentypen für Funktionen Θ 73

6 Überladene Funktionen und Operatoren 74

6.1 Inline-Funktionen Θ 74
6.2 Überladene Funktionen 75
6.2.1 Funktionen, die nicht überladen werden können 77
6.2.2 Regeln für die Auswahl einer passenden Funktion 77
6.3 Überladene Operatoren mit globalen Operatorfunktionen 81
6.3.1 Globale Operatorfunktionen 83
6.3.2 Die Ein- und Ausgabe von selbst definierten Datentypen 85
6.4 Referenztypen, Werte- und Referenzparameter 86
6.4.1 Werteparameter 86
6.4.2 Referenztypen 87
6.4.3 Referenzparameter 88
6.4.4 Referenzen als Rückgabetypen 89
6.4.5 Konstante Referenzparameter 91

7 Objektorientierte Programmierung 93

7.1 Klassen 93
7.1.1 Datenelemente und Elementfunktionen 94
7.1.2 Der Gültigkeitsbereich von Klassenelementen 97
7.1.3 Datenkapselung: Die Zugriffsrechte *private* und *public* 99
7.1.4 Der Aufruf von Elementfunktionen und der *this*-Zeiger 103
7.1.5 Konstruktoren und Destruktoren 104
7.1.6 OO Analyse und Design: Der Entwurf von Klassen 112
7.1.7 Klassendiagramme 115
7.2 Klassen als Datentypen 116
7.2.1 Der Standardkonstruktor 116
7.2.2 Elementinitialisierer 118
7.2.3 Initialisiererlisten 122
7.2.4 *friend*-Funktionen und –Klassen 125

7.2.5Überladene Operatoren mit Elementfunktionen ... 128
7.2.6Der Kopierkonstruktor ... 131
7.2.7Der Zuweisungsoperator = für Klassen ... 134
7.2.8Die Angaben =*delete* und =*default* ... 138
7.2.9Konvertierende und explizite Konstruktoren Θ ... 139
7.2.10Konversionsfunktionen mit und ohne *explicit* Θ ... 142
7.2.11*constexpr* Statische Klassenelemente ... 143
7.2.12Konstante Objekte und Elementfunktionen ... 145
7.2.13Funktionen als Objekte und Parameter mit *std::function* ... 147
7.2.14Delegierende Konstruktoren Θ ... 150
7.2.15Klassen und Header-Dateien ... 151
7.3 Vererbung und Komposition ... 152
7.3.1Die Elemente von abgeleiteten Klassen ... 152
7.3.2Zugriffsrechte auf die Elemente von Basisklassen ... 154
7.3.3Verdeckte Elemente ... 155
7.3.4Konstruktoren, Destruktoren und implizit erzeugte Funktionen ... 157
7.3.5OO Design: *public* Vererbung und „ist ein“-Beziehungen ... 162
7.3.6OO Design: Komposition und „hat ein“-Beziehungen ... 165
7.3.7Konversionen zwischen *public* abgeleiteten Klassen ... 166
7.3.8Mehrfachvererbung und virtuelle Basisklassen ... 169
7.4 Virtuelle Funktionen, späte Bindung und Polymorphie ... 173
7.4.1Der statische und der dynamische Datentyp ... 173
7.4.2Virtuelle Funktionen in C++03 ... 173
7.4.3Virtuelle Funktionen mit *override* in C++11 ... 174
7.4.4Die Implementierung von virtuellen Funktionen: *vptr* und *vtbl* ... 180
7.4.5Virtuelle Konstruktoren und Destruktoren ... 184
7.4.6Virtuelle Funktionen in Konstruktoren und Destruktoren ... 186
7.4.7OO-Design: Einsatzbereich und Test von virtuellen Funktionen ... 187
7.4.8OO-Design und Erweiterbarkeit ... 188
7.4.9Rein virtuelle Funktionen und abstrakte Basisklassen ... 190
7.4.10OO-Design: Virtuelle Funktionen und abstrakte Basisklassen ... 193
7.4.11Objektorientierte Programmierung: Zusammenfassung ... 194
7.5 R-Wert Referenzen und Move-Semantik ... 195
7.5.1R-Werte und R-Wert Referenzen ... 196
7.5.2move-Semantik und *std::move* ... 198
7.5.3Move-Semantik in der C++11 Standardbibliothek ... 202
7.5.4Move-Semantik für eigene Klassen ... 202
7.5.5Benchmarks ... 204

8 Namensbereiche ... 207

8.1 Die Definition von Namensbereichen ... 208
8.2 Die Verwendung von Namen aus Namensbereichen ... 210
8.3 Header-Dateien und Namensbereiche ... 212
8.4 Aliasnamen für Namensbereiche Θ ... 214

9 Exception-Handling ... 217

9.1 Die *try*-Anweisung ... 218
9.2 Exception-Handler und Exceptions der Standardbibliothek ... 221
9.3 *throw*-Ausdrücke und selbst definierte Exceptions ... 223
9.4 Fehler und Exceptions ... 228
9.5 Die Freigabe von Ressourcen bei Exceptions: RAII ... 229
9.6 Exceptions in Konstruktoren und Destruktoren ... 231
9.7 *noexcept* ... 235
9.8 Die Exception-Klasse *system_error* Θ ... 236

10 Containerklassen der C++-Standardbibliothek ... 239

10.1 Sequenzielle Container der Standardbibliothek ... 239
10.1.1Die Container-Klasse *vector* ... 239

10.1.2Iteratoren 242
10.1.3Geprüfte Iteratoren (Checked Iterators) 245
10.1.4Die bereichsbasierte *for*-Schleife 246
10.1.5Iteratoren und die Algorithmen der Standardbibliothek 248
10.1.6Die Speicherverwaltung bei Vektoren Θ 250
10.1.7Mehrdimensionale Vektoren Θ 251
10.1.8Die Container-Klassen *list* und *deque* 252
10.1.9Gemeinsamkeiten und Unterschiede der sequenziellen Container 253
10.1.10Die Container-Adapter *stack*, *queue* und *priority_queue* Θ 254
10.1.11Container mit Zeigern 255
10.1.12*std::array* - Array Container fester Größe Θ 255
10.2 Assoziative Container 256
10.2.1Die Container *set* und *multiset* 257
10.2.2Die Container *map* und *multimap* 257
10.2.3Iteratoren der assoziativen Container 259
10.2.4Ungeordnete Assoziative Container (Hash-Container) 260

11 Literaturverzeichnis 265

12 Index 267