SCHWARZ/WEIß

Detlef Wilkening

C++ KOMPENDIUM

PROFESSIONELL C++ PROGRAMMIEREN LERNEN

OHNE VORWISSEN LOSLEGEN

Eine umfassende Einführung in C++!

- Praktische Einführung in die C++ Programmierung
- Ausführliche Grundlagen des C++ Container-Framework
- Kontaktdaten-Verwaltung als Einstiegs-Programm

Inklusive eBook zum Download

{BMU VERLAG}

C++ Kompendium

1 Teil

Professionell C++ Programmieren lernen

Detlef Wilkening

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;
detaillierte bibliografische Informationen sind im Internet über http://dnb.d-nb.de abrufbar.

©2023 BMU Media GmbH
www.bmu-verlag.de
info@bmu-verlag.de

Lektorat: Lektormeister
Einbandgestaltung: Pro ebookcovers Angie
Druck und Bindung: Wydawnictwo Poligraf sp. zo.o. (Polen)

Taschenbuch-ISBN: 978-3-96645-041-6
Hardcover-ISBN: 978-3-96645-116-1
eBook-ISBN: 978-3-96645-040-9

Dieses Werk ist urheberrechtlich geschützt.
Alle Rechte (Übersetzung, Nachdruck und Vervielfältigung) vorbehalten. Kein Teil des Werks darf ohne schriftliche Genehmigung des Verlags in irgendeiner Form – auch nicht für Zwecke der Unterrichtsgestaltung- reproduziert, verarbeitet, vervielfältigt oder verbreitet werden.

Dieses Buch wurde mit größter Sorgfalt erstellt, ungeachtet dessen können weder Verlag noch Autor, Herausgeber oder Übersetzer für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären.

C++ Kompendium

1 Teil

1. Verzeichnisse

1.1 Inhaltsverzeichnis

1. Verzeichnisse 4

1.1 Inhaltsverzeichnis ... 4
1.2 Abbildungsverzeichnis ... 10

2. Vorwort 13

3. Einleitung 17

3.1 Die Geschichte von C++ ... 17
3.2 Die ISO C++ Standards ... 20
3.2.1 C++98 & C++03 ... 20
3.2.2 C++11 ... 21
3.2.3 C++14 ... 23
3.2.4 C++17 ... 23
3.2.5 C++20 ... 24
3.2.6 C++23 ... 24
3.3 Die Programmier-Paradigmen ... 24

4. Praxis 28

4.1 Programmier-Ablauf ... 28
4.1.1 Code beziehungsweise Quelltext ... 28
4.1.2 C++ Compiler ... 29
4.1.3 Dokumentation ... 30
4.1.4 Bibliotheken ... 31
4.1.5 Fazit ... 31
4.2 Editoren & Entwicklungs-Umgebungen ... 31
4.2.1 Editoren ... 31
4.2.2 Notepad++ ... 32
4.2.3 Microsoft Visual Code ... 33
4.2.4 IDE's ... 34
4.2.5 Microsoft Visual Studio Community 2022 ... 34
4.3 C++ Compiler ... 35
4.3.1 ISO C++ Konformität ... 37
4.3.2 Optimierungen ... 39
4.4 Bibliotheken ... 41
4.5 Boost ... 43
4.6 Dokumentation ... 45
4.7 Installation Microsoft Visual Studio ... 46
4.8 Microsoft Entwicklerdienste ... 50

5. Einführung 52

5.1 Das erste C++ Programm ... 53

5.2 Microsoft Visual Studio ... 57
5.2.1 Projekte und Workspaces ... 57
5.2.2 Microsoft Visual Studio ... 58
5.3 Compile-Fehler ... 74
5.4 Unser zweites C++ Programm ... 77
5.4.1 Microsoft Visual Studio ... 79
5.5 Nutzung von Namespaces ... 81
5.6 Internationalisierung & Zeichensätze ... 83
5.6.1 Zeichensätze beziehungsweise Zeichenkodierungen ... 84
5.6.2 Portabilität ... 85
5.6.3 Bearbeitung und Vergleich von Zeichen ... 87
5.6.4 Sichere Annahme über Zeichen ... 88
5.7 C++20 ... 89
5.8 Harter Programmabbruch ... 92
5.9 Debug- und Release-Modus ... 93
5.10 Undefined Behaviour ... 97
5.11 Aufgaben ... 103

6. Die grundlegenden Sprach-Elemente 104

6.1 Quelltext ... 104
6.1.1 Formloser Quelltext ... 104
6.1.2 Kommentare ... 105
6.1.3 Namen ... 106
6.2 Typen ... 106
6.2.1 Statische Typisierung ... 106
6.2.2 Optimierung ... 108
6.2.3 Allgemeines zu Typen ... 108
6.2.4 Elementare Typen ... 109
6.3 Literale ... 118
6.3.1 Wahrheitswerte ... 118
6.3.2 Ganz-Zahlen ... 118
6.3.3 Fließkomma-Zahlen ... 119
6.3.4 Zeichen ... 119
6.3.5 Zeichenketten-Konstanten und String-Literale ... 120
6.4 Variablen ... 122
6.4.1 Variablen-Arten ... 123
6.4.2 Globale Variablen ... 126
6.4.3 Variable Definitionen ... 127
6.5 Konstanten ... 141
6.5.1 Lokale und globale Konstanten ... 144
6.5.2 Kombination von „const“, „auto“ und „decltype“ ... 145
6.5.3 Linksbindend, außer ... 146
6.5.4 Const-Korrektheit ... 148
6.6 Operatoren ... 149
6.6.1 Zuweisungs-Operatoren ... 150
6.6.2 Mathematische Operatoren ... 154
6.6.3 Vergleichs Operatoren ... 158
6.6.4 Logische Operatoren ... 160
6.6.5 Prä- und Post-Inkrement und -Dekrement Operatoren ... 164
6.6.6 Fragezeichen-Operator ... 166
6.6.7 Bit-Operatoren ... 167

6.7 Aufgaben ... 171
6.7.1 Grundrechen-Arten ... 171
6.7.2 Increment-Operator ... 171

7. Kontrollstrukturen 172

7.1 Bedingter-Kontrollfluss – „if“ & „else“ ... 172
7.1.1 Einfachste Form ... 172
7.1.2 Blöcke für mehrere Anweisungen ... 175
7.1.3 If-Else Anweisung ... 176
7.1.4 Mehrfach-Verzweigungen mit If-Else-If ... 177
7.1.5 If mit Initialisierungs-Teil ... 179
7.2 Mehrfach-Verzweigung – „switch“ ... 180
7.2.1 Ein größeres Beispiel ... 184
7.2.2 Switch-Anweisung mit Initialisierung-Teil ... 185
7.2.3 Break, Durchfallen und Fallthrough ... 185
7.2.4 Warum eine Switch-Anweisung? ... 187
7.3 For-Schleife ... 188
7.3.1 Zählschleife ... 189
7.3.2 Range-basierte For-Schleife ... 191
7.4 While-Schleife ... 193
7.5 Do-Schleife ... 194
7.6 Break- und Continue-Anweisungen ... 195
7.6.1 Break-Anweisung ... 195
7.6.2 Continue-Anweisung ... 197
7.7 Schleife mit Ausgang in der Mitte ... 199
7.8 „goto“ und Labels ... 201
7.8.1 Syntax ... 202
7.8.2 Verlassen mehrerer ineinander verschachtelter Schleifen ... 202
7.9 Aufgaben ... 203
7.9.1 Schleifen-Varianten ... 203
7.9.2 Teilbar? ... 204
7.9.3 Lesbare Zahlen ... 204
7.9.4 Zahlen-Liste ... 205
7.9.5 Multiplikations-Matrix ... 205

8. Ein- und Ausgabe 206

8.1 Ausgabe ... 206
8.1.1 Manipulatoren ... 208
8.1.2 Pufferung ... 208
8.1.3 Boolesche Ausgaben ... 211
8.1.4 Formatierungen ... 212
8.2 Fehlschläge ... 213
8.2.1 Etwas Pragmatismus ... 215
8.3 Eingabe ... 216
8.3.1 Fehlschläge ... 218
8.3.2 Stream leeren ... 220
8.3.3 Zurück zum Einlesen ... 225
8.3.4 Programm-Fluss und -Kontrolle ... 225
8.4 Weiteres ... 228
8.4.1 Weitere Streams ... 228

8.4.2 Exceptions statt Fehler-Abfrage ... 228
8.4.3 *FAIL* Status selber setzen ... 229
8.5 Aufgaben ... 229
8.5.1 Summe ... 229
8.5.2 Lesbare Zahlen 2 ... 230
8.5.3 Multiplikations-Matrix 2 ... 231

9. Texte 232

9.1 Der Standard-String „std::string“ ... 233
9.1.1 String-Erzeugung ... 233
9.1.2 Der Typ „std::string::size_type“ ... 236
9.1.3 Länge ... 237
9.1.4 Ein- und Ausgabe ... 238
9.1.5 String-Verkettungen ... 240
9.1.6 Vergleiche ... 241
9.1.7 Index-Zugriff auf einzelne Zeichen ... 243
9.1.8 Teil-Strings ... 244
9.1.9 Suchen und Ersetzen ... 245
9.1.10 Löschen ... 246
9.2 String-Views ... 247
9.3 String-Wandlungen ... 250
9.4 Reguläre Ausdrücke ... 252
9.5 Aufgaben ... 255
9.5.1 Hallo <Person> ... 255
9.5.2 Leerzeichen zählen ... 255
9.5.3 String-Analyse ... 256
9.5.4 Zahlen-Palindrom ... 256

10. Weiteres zum Typ-System 258

10.1 Typ-Aliase ... 258
10.1.1 Typ-Namen vereinfachen ... 259
10.1.2 Konkrete Typen verstecken und semantische Typ-Namen erzeugen ... 260
10.1.3 Leichte Änderbarkeit von Typen ... 261
10.2 Referenzen ... 262
10.2.1 Anwendungen von Referenzen ... 264
10.2.2 Const-Referenzen ... 266
10.2.3 Referenzen mit „auto“ ... 266
10.2.4 Lebensdauer des referenzierten Objekts ... 267
10.2.5 R-Value Referenzen ... 268
10.2.6 Forward-Referenzen ... 269
10.3 Typ-Konvertierungen ... 270
10.3.1 Implizite Typ-Konvertierungen ... 270
10.3.2 Explizite Typ-Konvertierungen ... 274
10.4 Aufzählungs-Typen ... 277
10.4.1 Grundlagen ... 277
10.4.2 Enums basieren auf Integern ... 280
10.5 Aufgaben ... 284
10.5.1 Typ-Aliase ... 284
10.5.2 Referenzen ... 284
10.5.3 Rechner mit Enums ... 285

11. Funktionen 286

11.1 Einführung ... 286
11.1.1 Funktions-Definitionen ... 287
11.1.2 Funktions-Aufrufe ... 289
11.2 Funktions-Deklarationen ... 290
11.3 Freie Funktionen ... 293
11.4 Parameter und Argumente ... 295
11.4.1 call-by-value ... 296
11.4.2 call-by-reference ... 297
11.4.3 call-by-const-reference ... 298
11.4.4 Wann, was und warum? ... 299
11.5 Funktions-Rückgaben ... 302
11.5.1 Erweiterte Syntaxen für Funktions-Rückgaben ... 303
11.5.2 Nicht zwingende Nutzung der Rückgabe ... 306
11.5.3 Direkte Nutzung der Rückgabe ... 307
11.5.4 Const Rückgaben ... 308
11.5.5 Referenz Rückgaben ... 309
11.5.6 Rückgabe neuer Objekte ... 311
11.6 Default-Argumente ... 312
11.7 Funktions-Überladung ... 314
11.7.1 Flexibilität ... 315
11.7.2 Konvertierungen & Mehrdeutigkeiten ... 317
11.7.3 Überladen bei Referenzen mit und ohne „const“ ... 324
11.8 Rekursion ... 327
11.8.1 Lokale Variablen beim Funktions-Aufruf ... 327
11.8.2 Rekursion ... 331
11.9 Constexpr & consteval Funktionen ... 333
11.9.1 Constexpr Funktionen ... 333
11.9.2 Consteval Funktionen ... 337
11.10 Funktions-Templates ... 338
11.10.1 Templates mit „auto“ ... 338
11.10.2 Explizite Templates ... 339
11.10.3 Explizite Template-Argument Angabe ... 341
11.10.4 Templates überladen ... 342
11.10.5 Template Wert-Parameter ... 343
11.10.6 Concepts ... 344
11.10.7 Template Parameter-Packs ... 348
11.10.8 Fold-Expressions ... 350
11.11 Aufgaben ... 350
11.11.1 Swap-Funktion ... 350
11.11.2 Generische Swap-Funktion ... 351
11.11.3 String-Länge setzen ... 351
11.11.4 Fakultät ... 352
11.11.5 Rekursive Multiplikation ... 352
11.11.6 Zahlen-Wandlung ... 353
11.11.7 Formatierungs-Funktion ... 353

12. Container, Algorithmen & Lambdas 356

12.1 Einführung ... 356

12.2 Vektoren ... 357
12.2.1 Erste Nutzung ... 357
12.2.2 std::vector<>::size_type ... 359
12.2.3 Wahlfreier Zugriff ... 359
12.2.4 Noch mal der Typ der Zähl-Variablen ... 362
12.2.5 Einfügen und Löschen ... 363
12.2.6 Suchen ... 366
12.2.7 Ein Vektor von Strings ... 367
12.3 Listen ... 368
12.4 Iteratoren ... 371
12.5 Range-basierte For-Schleife ... 378
12.5.1 Referenzen im Kontext der range-basierten For-Schleife ... 379
12.6 Arrays ... 383
12.7 Sets ... 385
12.7.1 Sortiertes Set ... 386
12.7.2 Mehrfache Elemente und Element-Identität ... 388
12.7.3 Set-Sortierung ... 389
12.7.4 Zeichen- und String-Sortierung ... 389
12.7.5 Eigene Sortierung ... 391
12.7.6 Implementierung des sortierten Sets ... 393
12.7.7 Suchen im sortierten Set ... 395
12.7.8 Unsortierte Sets ... 399
12.7.9 Allgemeines zu Sets ... 407
12.8 Maps ... 407
12.8.1 Sortierte Map ... 407
12.8.2 Suchen in der Map ... 411
12.8.3 Zugriff in die Map mit dem Index-Operator ... 413
12.8.4 Unsortierte Map ... 416
12.9 Aufgaben ... 418
12.9.1 Lesbare Zahlen 3 ... 418
12.9.2 Hallo <Personen> ... 419
12.9.3 Leerzeichen zählen 2 ... 419
12.9.4 String-Analyse 2 ... 420
12.9.5 Algorithmen Anwendung ... 420

13. Erste größere Beispiele 422

13.1 Kontaktdaten-Verwaltung 1 ... 422
13.1.1 Ziel ... 422
13.1.2 Umsetzung ... 424
13.1.3 Auflistung der kompletten Kontaktdaten ... 427
13.1.4 Suchen ... 427
13.1.5 Zusammenfassung ... 428
13.2 Kontaktdaten-Verwaltung 2 ... 429
13.2.1 Ziel ... 429
13.2.2 Umsetzung ... 431

14. Fazit 447

15. Stichwortverzeichnis 450

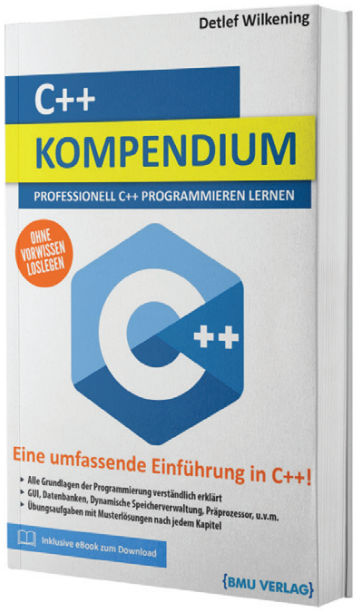
1.2 Abbildungsverzeichnis

Abb. 4-2 Notepad++ mit C++ „Hallo Welt“ Programm 32
Abb. 4-3 Microsoft Visual Code mit C++ „Hallo Welt“ Programm 33
Abb. 4-4 Microsoft Visual Studio Community mit C++ „Hallo Welt“ Programm 35
Abb. 4-5 Optimierung 39
Abb. 4-6 Compilierung mit Bibliothek 41
Abb. 4-7 Microsoft Web-Seite für das Microsoft Visual Studio Community 47
Abb. 4-8 : Download des Microsoft Visual Studio Community Installers 48
Abb. 4-9 Microsoft Visual Studio Community Installer in Arbeit 48
Abb. 4-10 Microsoft Visual Studio Installations Auswahl-Dialog 49
Abb. 4-11 Microsoft Visual Studio Community Installer in Arbeit 49
Abb. 4-12 Neustart des Rechners am Ende der Installation 50
Abb. 4-13 Anmelde-Dialog für die Microsoft Entwicklerdienste 50
Abb. 5-1 : Programmier-Ablauf 52
Abb. 5-2 Startbildschirm des Microsoft Visual Studios 58
Abb. 5-4 Projekt-Erstellungs-Dialog 60
Abb. 5-5 Dialog zum Anlegen einer leeren Projektmappe 61
Abb. 5-6 Leere Projektmappe im Microsoft Visual Studio 61
Abb. 5-7 Neue Projektmappe auf Ebene des Dateisystems 62
Abb. 5-8 : Neues Projekt in der Projektmappe anlegen 62
Abb. 5-9 Hinzufügen eines neuen leeren Projekts 63
Abb. 5-10 Dialog zum Anlegen eines leeren Projekts 64
Abb. 5-11 Projektmappe mit unserem ersten Projekt 64
Abb. 5-12 Filter im Projekt löschen 65
Abb. 5-13 Das erste Projekt im Dateisystem 66
Abb. 5-14 Neues Element dem Projekt hinzufügen 67
Abb. 5-15 Dialog zum Hinzufügen von neuen Elementen 67
Abb. 5-16 Projekt mit C++ Datei „main.cpp“ 68
Abb. 5-17 C++ Datei im Dateisystem 68
Abb. 5-18 Unser erstes C++ Programm im Editor 69
Abb. 5-19 Projekt erstellen 69
Abb. 5-20 Compiler-Ausgabe im Ausgabe View 70
Abb. 5-21 Starten unseres Programms 70
Abb. 5-22 Unser erstes Programm läuft in der Visual Studio Konsole 71
Abb. 5-23 Debug-Verzeichnis im Projekt-Verzeichnis 71
Abb. 5-24 Debug-Verzeichnis im Projektmappen-Verzeichnis 72
Abb. 5-25 Starten der Windows Eingabeaufforderung 73
Abb. 5-26 Unser Programm in der Windows Eingabeaufforderung 74
Abb. 5-27 Syntax-Fehler im Microsoft Visual Studio 74
Abb. 5-28 Fehler Tooltip im Microsoft Visual Studio 75
Abb. 5-29 Fehler im Fehlerlisten-View 75
Abb. 5-30 Fehler im Ausgabe-View 76
Abb. 5-31 Der eigentliche Fehler liegt woanders 76
Abb. 5-32 Das zweite Projekt in der Projektmappe 79

Abb. 5-33 Startprojekt definieren ... 80
Abb. 5-34 Das Startprojekt wird fett dargestellt ... 81
Abb. 5-35 Umlaute Beispiel im Microsoft Visual Studio ... 87
Abb. 5-36 Umlaute Beispiel in der Visual Studio Konsole ... 87
Abb. 5-37 Projekteigenschaften im Kontext-Menü des Projektmappen-Explorers ... 90
Abb. 5-38 Projekteigenschaften auf C++20 stellen ... 91
Abb. 5-39 Endlos-Programm in der Visual Studio Konsole beenden ... 92
Abb. 5-40 Endlos-Programm in der Windows Eingabeaufforderung beenden ... 93
Abb. 5-41 Modi im Microsoft Visual Studio ... 94
Abb. 5-43 Konfiguration der C++ Optimierungen im Release-Modus ... 96
Abb. 5-44 Release Verzeichnisse im Dateisystem ... 96
Abb. 5-45 Zugriff auf ein nicht existentes Zeichen ... 99
Abb. 5-46 Zugriff auf ein nicht existentes Zeichen im Debug-Modus ... 101
Abb. 5-47 Harter Programmabbruch durch Windows ... 102
Abb. 6-1 Variablen existieren im Speicher (RAM) des Computers ... 122
Abb. 9-2 Ein String-View ist eine Sicht auf eine Zeichenkette ... 248
Abb. 11-1 Ablauf der Funktion „f“ ... 330
Abb. 11-2 Rekursive Summen-Berechnung ... 332
Abb. 12-1 Speichermodell eines Vektors ... 357
Abb. 12-2 Random-Access Zugriff bei einem Vektor ... 360
Abb. 12-3 Hinten Einfügen und Löschen im Vektor ... 363
Abb. 12-4 Vorne Einfügen und Löschen im Vektor ... 365
Abb. 12-5 Speichermodell einer doppelt verketteten Liste ... 368
Abb. 12-6 Einfügen in der doppelt-verketteten Liste ... 369
Abb. 12-7 Löschen in der doppelt-verketteten Liste ... 369
Abb. 12-8 Konzept der Iteratoren am Beispiel „std::vector“ ... 372
Abb. 12-9 Speichermodell eines Arrays ... 383
Abb. 12-10 Speichermodell eines sortierten Sets ... 393
Abb. 12-11 Einfügen im sortierten Set ... 394
Abb. 12-12 Entstehung eines entarteten Baums ... 395
Abb. 12-13 Suchen im sequentiellen Container ... 396
Abb. 12-14 Suchen im sortierten Set ... 397
Abb. 12-15 Speichermodell eines unsortierten Sets ... 400
Abb. 12-16 Einfügen von „Ada“ in das unsortierte Set ... 402
Abb. 12-17 Einfügen mit Kollision in das unsortierte Set ... 404
Abb. 12-18 Ein „entartetes“ unsortiertes Set ... 404
Abb. 12-19 Reorganisation eines unsortierten Sets ... 405
Abb. 12-20 Speichermodell einer sortierten Map ... 408
Abb. 12-21 Speichermodell einer unsortierten Map ... 417

Downloadhinweis

Alle Programmcodes aus diesem Buch sind zum Download verfügbar. Dadurch müssen Sie sie nicht abtippen:
https://bmu-verlag.de/books/cpp-kompendium/



Außerdem erhalten Sie die eBook Ausgabe zum Buch im PDF Format kostenlos auf unserer Website:

https://bmu-verlag.de/books/cpp-kompendium/
Downloadcode: siehe Kapitel 14

Kapitel 2
Vorwort

Die vielleicht wichtigste Frage zu diesem Buch ist: Für wen ist das Buch geschrieben? Kurz gesagt: Es ist sowohl für Einsteiger[1], für Umsteiger als auch für Fortgeschrittene – mit dem Focus auf die Einsteiger.

Die primäre Zielgruppe des Buchs sind Einsteiger in das Thema Programmieren. Interessierte, die sich grob mit Computern auskennen, aber nur wenige oder noch gar keine Programmiererfahrung haben und sich jetzt detaillierter damit auseinandersetzen möchten. Daher habe ich mich bemüht, bei allen Themen keine Kenntnisse vorauszusetzen und gerade die praktischen Aspekte im Umgang mit dem Compiler und der IDE, die für Anfänger oft eine große Hürde sind, am Anfang angemessen zu betrachten. Außerdem stellen alle Code-Beispiele immer nur ein einziges neues Feature vor oder decken nur ein neues Detail ab. Ich kann mich noch gut erinnern, wie verloren ich mich gefühlt habe, wenn ein Beispiel viele neue Aspekte auf einmal abgedeckt hat und man den Wald vor lauter Bäumen nicht gesehen hat. Zu guter Letzt sind alle größeren Code-Beispiele vollständig – so enthalten sie zum Beispiel auch immer alle „Includes“. Für jeden C++ Entwickler mit nur etwas Erfahrung ist dies nicht notwendig, da die „Includes“ zu den absoluten Basics gehören und schnell in Fleisch und Blut übergehen. Aber Anfänger stolpern häufig über solche Kleinigkeiten und wissen sich noch nicht zu helfen.

---

[1] Auch wenn ich von Einsteigern spreche, so sind natürlich auch Einsteigerinnen gemeint. Für den Lesefluss ist es einfach angenehmer, hier keine Auflistungen der Geschlechter vorzunehmen – gemeint sind natürlich alle. Bei der Nutzung der männlichen Form habe ich mich an der gebräuchlichen Nutzung in der deutschen Sprache orientiert – dies soll keine Bevorzugung oder Respektlosigkeit sein. Ich hoffe, Sie verstehen das.

Auch wenn dies eine einsteigerfreundliche Ausrichtung ist – das Buch ist ebenfalls für fortgeschrittene Entwickler und Umsteiger von anderen Sprachen, wie zum Beispiel C, Java oder Python, gedacht. Auch für sie ist eine systematische Einführung in die Sprachfeatures wichtig und hilfreich. Auch sie profitieren davon, dass ich die praktischen Seiten abdecke, die oft vernachlässigt werden und auch bei einem Umstieg ein Einstiegshindernis darstellen. Profitieren werden Umsteiger sicher von dem pragmatischen Vorgehen des Buchs. Die Themen werden kompakt vorgestellt und mit Beispielen behandelt – ohne dass ich mich in langatmigen Wiederholungen und Erklärungen verliere.

Ebenso kann selbst der erfahrenere Entwickler sicher noch manche Perle finden, wenn er sich auf die Reise durch modernes C++ einlässt. Gerade hier ist es häufig so, dass durch die tägliche Arbeit mit einer Programmiersprache die Kernelemente sehr gut bekannt sind, aber viele Details und Neuentwicklungen nie vertieft worden sind. Besonders die Neuigkeiten der letzten Standards C++17 und C++20 sind oft nur wenig verbreitet.

Meine Intention war es, dass der Leser das Buch von vorne nach hinten durcharbeitet. Alle Kapitel basieren auf dem Wissen, das in den vorhergehenden Kapiteln aufgebaut wurde. Natürlich kann sich ein Entwickler mit Vorkenntnissen auch einzelne Kapitel herauspicken und das Buch sprunghaft lesen. Für einen Anfänger oder Umsteiger bietet sich aber das lineare Durcharbeiten an. Auch die Beispiele und Aufgaben orientieren sich an diesem Aufbau. So tauchen einige Aufgaben mehrfach auf und entwickeln sich weiter. Entweder können sie mit fortschreitendem Wissen einfacher gelöst werden oder sie werden umfangreicher. Zum Beispiel ist beim ersten Tic-Tac-Toe die Zuordnung der Spieler noch genau festgelegt, in einer späteren Version können diese frei gewählt werden. Oder zum Beispiel zieht sich eine Kontaktdaten-Verwaltung durch das gesamte Buch, von einer einfachen Kommandozeilen-Anwendung mit festen Einträgen, in der nur nach Namen gesucht werden kann, bis hin zu einer Verwaltung mit Editier- und Speichermöglichkeiten. In einem folgenden Buch wird die Kontaktdaten-Verwaltung unter anderem mit einer grafischer Oberfläche und Datenbank versehen. Schritt

für Schritt, Kapitel für Kapitel entwickeln sich Ihre Fähigkeiten, C++ Programme zu implementieren.

Leider ist C++ mittlerweile so umfangreich geworden, dass eine halbwegs angemessene Einführung in die Sprache nicht mehr in ein Buch passt. Das ist auch hier der Fall. Darum enthält dieses Buch viele fortgeschrittene Themen nicht. In Kürze wird es einen zweiten Band geben, der in die fehlenden Inhalte einführen wird.

Das Buch fokussiert sich in der praktischen Ausrichtung auf die C++ Entwicklung unter Windows mit dem Microsoft Visual Studio. Es würde den Rahmen jedes Buches sprengen, alle möglichen Plattformen und Entwicklungs-Umgebungen abdecken zu wollen. Ich habe mich für Windows entschieden, da es das mit Abstand verbreitetste Betriebssystem ist. Und das Microsoft Visual Studio ist eine sehr gute Entwicklungsumgebung für C++, die es kostenlos bei Microsoft gibt. Trotzdem sind alle Aspekte der Sprache C++ im Buch unabhängig von Windows und dem Microsoft Visual Studio beschrieben. Nur der praktische Einstieg und einige Ausflüge in die Nutzung der Entwicklungsumgebung sind Windows beziehungsweise Microsoft Visual Studio bezogen.

Ich widme das Buch meiner wunderbaren Partnerin Marion Bylaitis, die auf vielfältigste Weise mein Leben unendlich bereichert und die Sonne für mich scheinen lässt. Ohne sie wäre das Buch sicher nicht entstanden, da sie mich immer wieder motiviert hat, und auch dann an mich geglaubt hat, wenn meine Lust zum Schreiben sehr gering war. Sie hat mir zudem viele neue Seiten des Lebens außerhalb der Informatik eröffnet und zeigt mir immer wieder, dass es andere und auch viel wichtigere Dinge auf der Welt gibt als C++, selbst wenn mir das zuweilen schwer zu glauben fiel – und das natürlich nicht für dieses Buch gilt.

Kein Buch ohne Vorwort, und kein Vorwort ohne eine Danksagung. Auch ich mache da keine Ausnahme. Zuallererst möchte ich mich bei meiner Partnerin Marion Bylaitis bedanken, die viele Stunden auf mich verzichten und manches Mal ihre Bedürfnisse hinten anstellen musste, damit ich das Buch in Worte fassen konnte. Gleichfalls bedanke ich mich auch bei meinen Söhnen Jan und Nils Wilkening, die zwar

mittlerweile schon erwachsen sind – sicher aber trotzdem gerne mehr Zeit mit mir verbracht hätten. Mein Dank gilt auch meinen Freunden Wolfgang Keller, Sven Johannsen und Christian Bryllok, die Entwürfe des Buches gelesen und es mit ihren Anmerkungen vorangebracht haben. Wenn das Buch trotzdem noch Fehler enthält, so sind diese allein meine Schuld. Eine große Hilfe waren aber auch all die Studenten der FH Aachen, Kollegen und Auszubildenden der Firma DSA, die mich und meine C++ Trainings in den letzten 25 Jahren ertragen mussten, und deren Fragen mir klar gemacht haben, wo ein Anfänger seine Probleme hat. Viel für mein Wissen und Verständnis von C++ verdanke ich auch den Mitgliedern der C++ User-Groups Aachen, Düsseldorf und Köln, deren Vorträge und Diskussionen extrem lehrreich und bereichernd waren. Und natürlich möchte ich mich auch bei den Mitarbeitern des BMU Verlags bedanken, die das Projekt mit so viel Geduld begleitet haben und nie das Vertrauen verloren haben, auch wenn ich manchen versprochenen Termin gerissen habe.

Kapitel 3
Einleitung

3.1 Die Geschichte von C++

In seinem Buch „Design and Evolution of C++“ beschreibt Bjarne Stroustrup detailliert die Entstehung von C++. Während seiner Doktorzeit an der Universität Cambridge kam er mit der objekt-orientierten Programmiersprache Simula in Berührung. Ihn begeisterten damals auf Anhieb die objekt-orientierten Ansätze von Simula – trotzdem wurde er damit nicht glücklich. Denn obwohl seine Programme auf einem Großrechner liefen, waren sie extrem langsam.

> Simula wurde von Ole-Johan Dahl und Kristen Nygaard in den 60er Jahren am norwegischen Rechenzentrum der Universität Oslo entwickelt. Die Sprache gilt als erste objekt-orientierte Programmiersprache. Sie enthält schon viele wichtige moderne Konzepte der Objekt-Orientierung, wie zum Beispiel Datenabstraktion, Modularisierung und Klassen.

Nach Abschluss seiner Doktorarbeit wechselte Bjarne Stroustrup zur den AT&T Bell Labs und lernte hier die Programmiersprache C kennen. C ist das genaue Gegenteil von Simula. Während Simula auf abstrakte Konzepte setzt, ist C sehr hardwarenah.

> C wurde von Dennis Ritchie von 1969 bis 1973 an den AT&T Bell Labs entwickelt mit dem Ziel, eine Programmiersprache zu haben, mit der man ein Betriebssystem schreiben kann. Bis zu diesem Zeitpunkt waren alle Betriebssysteme in Assembler geschrieben, da damals nur Assembler überhaupt die Möglichkeiten bot, die man für das Schreiben eines Betriebssystems benötigte – zum Beispiel genaue Kontrolle über das Speicherlayout oder höchste Per-

formance. Leider ist Assembler Code extrem maschinenspezifisch. Will man ein Assembler Programm auf einen anderen Prozessor portieren, so muss man quasi das gesamte Programm neu schreiben.

1969 wurde das Betriebssystem Unix an den AT&T Bell Labs von Ken Thompson und Dennis Ritchie entwickelt, und dabei vollständig in Assembler programmiert. Durch die Implementierung in Assembler war es nur schwer möglich, Unix auf andere Hardware-Architekturen zu portieren. Darum entwickelte Dennis Ritchie die Programmiersprache C mit eben dem Ziel, das Betriebssystem Unix in einer Hochsprache schreiben zu können. 1973 war die Programmiersprache C dann so weit, dass er zusammen mit Ken Thompson den Unix-Kernel für die PDP-11 in C neu schreiben konnte.

C zeichnet sich durch sehr wenige, aber sehr maschinen-nahe Konzepte aus – allen voran zum Beispiel die Orientierung der Typ-Größen an der Hardware, aber auch Zeiger, Unions und Bitfelder lassen sich eins zu eins auf Hardware abbilden. Die Wahl der Konzepte von C ist Dennis Ritchie damals sehr gut gelungen – sonst wäre sie, wie zum Beispiel Simula, sicher schon längst ausgestorben. Selbst wenn C heute sehr umstritten ist, so ist auch noch heute C eine der meistbenutzten Programmiersprachen der Welt und wird gerade in der Embedded- und System-Entwicklung sehr häufig eingesetzt. Das liegt daran, dass die Konzepte von C die Maschine sehr gut abbilden und der Programmierer damit sehr viel Kontrolle über sein Programm hat – und dies ist in vielen Bereichen sehr wichtig und von hohem Vorteil.

Zurück zu Bjarne Stroustrup: Er lernte also C als eine Sprache kennen, die auf höchste Effizienz und Kontrolle ausgelegt war. Und er war nach seinen Erfahrungen mit Simula überrascht, dass man in einer Hochsprache Programme schreiben konnte, die selbst auf einem kleinen Computer performant liefen. Und wir reden hier über die späten 70er Jahre – und da waren die Computer noch viel langsamer als heute.

Man kann sich denken, was passiert ist: Bjarne Stroustrup wollte gerne beide Philosophien vereinen: die Abstraktionen und Möglichkeiten von Simula mit der Effizienz und Kontrolle von C. Performante Programme – geschrieben mit abstrakten Konzepten –, das war sein Ziel. Und so startete er im Mai 1979 die entsprechende Entwicklung. Den Namen „C++“ gab es damals noch nicht. Bjarne Stroustrup nannte seine Programmiersprache einfach „C mit Klassen“, denn viel mehr war es am Anfang nicht. Er erweiterte einfach nur die Programmiersprache C um die Klassen- und Vererbungs-Konzepte, die er aus Simula kannte – und fügte noch eine strengere Typisierung, Inlining sowie Default-Argumente hinzu.

Innerhalb kurzer Zeit wurde „C mit Klassen“ nicht nur von Bjarne Stroustrup produktiv eingesetzt, sondern auch von anderen Abteilungen bei den AT&T Bell Labs. Irgendwann fand „C mit Klassen“ dann den Weg in die Öffentlichkeit und verbreitete sich immer mehr. 1983 wurde „C mit Klassen“ in „C++“ umbenannt und die Programmiersprache um Operator-Überladung, virtuelle Funktionen und vieles mehr erweitert.

1985 war dann ein wichtiger Meilenstein für C++: Bjarne Stroustrup schrieb sein Buch „The C++ Programming Language“, das bis 1990 die inoffizielle Referenz für die Programmiersprache C++ war. Zudem wurde der erste kommerzielle C++ Compiler („Cfront 1.0“) vorgestellt. C++ hatte den Weg in die breite Öffentlichkeit gefunden. Außerdem – auch wenn es noch für 6 Jahre nicht relevant für C++ war – wurde 1985 die ISO Standardisierungsgruppe „ISO/IEC JTC 1/SC 22“ gegründet, die sich mit der ISO Standardisierung von Programmiersprachen beschäftigen sollte. Die Working Group 21 („ISO/IEC JTC 1/SC 22/WG 21“) – gegründet 1991 – sollte später für C++ zuständig werden.

1989 wurde C++ 2.0 mit vielen neuen Sprach-Features publiziert und die zweite Version des C++ Compilers „Cfront“ wurde ausgeliefert. 1990 erschien dann das Buch „The Annotated C++ Reference Manual“, geschrieben von Margaret A. Ellis und Bjarne Stroustrup. So wie 1985 das Buch „The C++ Programming Language“ die inoffizielle Referenz für C++ wurde, so übernahm jetzt „The Annotated C++ Reference Manual“ diese Rolle. Außerdem wurde 1990 das ANSI C++ Komitee gegründet

und 1991 mit der Working Group 21 das ISO C++ Standardisierungs-Komitee. Spätestens jetzt betrat C++ die Bühne der großen und wichtigsten Programmiersprachen.

3.2 Die ISO C++ Standards

1991 begann die ISO Standardisierung von C++ mit Gründung der „ISO/IEC JTC 1/SC 22/WG 21". Das klingt vielleicht ganz harmlos, ist aber für C++ von immenser Bedeutung: Die meisten Programmiersprachen wurden und werden von einer Universität oder einer großen Firma entwickelt, zum Beispiel „Java" lange Zeit von Oracle (früher von Sun), „C#" von Microsoft, „Go" von Google, „Swift" von Apple, oder „Kotlin" von JetBrains. In gewisser Weise *„gehören"* die Sprachen damit diesen Universitäten oder Firmen – zumindest bestimmen sie die Entwicklung der Programmiersprache sehr stark mit und stellen oft die einzige Implementierung. C++ dagegen ist ein offener Standard, an dem ganz viele Universitäten, Firmen und auch Privatpersonen mitwirken – hier arbeiten Microsoft, Apple, Google, Adobe und viele mehr gemeinsam an einem Ziel und es gibt viele Implementierungen. C++ gehört niemandem, sondern uns allen.

3.2.1 C++98 & C++03

Die Standardisierung setzte 1991 auf das „The Annotated C++ Reference Manual" auf und steckte sich das Ziel, innerhalb weniger Jahre den ersten ISO C++ Standard zu publizieren. Wie so oft machen Menschen Pläne, um Gott zu amüsieren, und die Wirklichkeit kommt ganz anders. So war es auch hier: Kurz vor „Fertigstellung" des Standards präsentierte Alexander Stepanov im November 1993 seine bei HP mit Meng Lee entwickelten Ideen für ein Container- und Algorithmen-Framework dem C++ Standardisierungs-Komitee. Das Standardisierungs-Komitee war sehr begeistert von den Ideen und beschloss, ein solches Framework in die Bibliothek von C++ aufzunehmen. Dies führte zu vielen Verzögerungen, da hierfür auch die Programmiersprache C++ erweitert werden musste. Das Ergebnis war dann, dass erst im September 1998 der erste

C++ Standard erschien: „ISO/IEC 14882:1998“ – meist kurz C++98 genannt.

C++03 vom November 1993 („ISO/IEC 14882:2003“) war im Prinzip nur eine technische Korrektur. C++03 enthielt gegenüber C++98 nur ein neues, sehr kleines Sprach-Feature („Wert-Initialisierungen“) und bestand ansonsten nur aus Klarstellungen, Fehlerberichtigungen und Umformulierungen. Beide Standards sind im Prinzip identisch.

3.2.2 C++11

Nach C++98 wurde das „Leben“ für C++ lange Zeit sehr schwierig. Die Programmiersprache war in der ersten Version fertig und das Standardisierungs-Komitee pausierte. Viele Firmen widmeten sich anderen Themen statt der Weiterentwicklung von C++ und der zugehörigen Infrastruktur. Die Programmiersprache Java erlebte ihren großen Aufschwung, bei vielen Firmen wurden die Entwickler-Kapazitäten von C++ abgezogen und zum Beispiel in Java oder C# gesteckt, oder in grafische Assistenten und hübsche Web-Seiten. Es war nicht so, dass gar nichts mehr im Ökosystem C++ passierte, aber es war vergleichsweise wenig. Zu wenig, um das langfristige Überleben zu sichern. C++ geriet gegenüber anderen Programmiersprachen immer mehr ins Hintertreffen und schien dem langsamen Tod gewidmet zu sein.

Was genau die Änderung herbeiführte, ist auch heute noch unklar und wird es wohl immer bleiben. In der Mitte des ersten Jahrzehnts dieses Jahrhunderts erkannte man jedenfalls, dass die Programmierwelt viele Probleme bereithielt, für die andere Programmiersprachen keine angemessenen Lösungen enthielten, und man besann sich wieder auf C++. Ein Auslöser war sicher das Ende der jährlichen Steigerungsraten bei den Taktfrequenzen der Prozessoren.

Seit rund 1975 verdoppelte sich alle ca. 2 Jahre die Taktfrequenz der Prozessoren (analog zum Mooreschen Gesetz). Der normale Computer von 1992 war also rund doppelt so schnell wie der normale Computer von 1990, der von 1994 wieder doppelt so schnell wie der von 1992 und

damit viermal so schnell wie der von 1990. Als Programmierer konnte man also ein Programm schreiben, das zu langsam war, und einfach etwas warten, denn die schnelleren Computer lösten das Problem von alleine. Ich selbst habe das mehrere Male in meinem Leben so machen können. Seit rund 2002 ist diese Steigerung vorbei. Wir sind im Prozessorbau an physikalische Grenzen gekommen. Aus dem Faktor 2 wurde seitdem eine Steigerung von nur noch rund 5% pro Jahr. Herb Sutter kommentierte 2005 diese Entwicklung wie folgt: „Free lunch is over." Wir müssen als Programmierer wieder selber etwas für die Performance machen. Und C++, in der Nachfolge von C, wurde entwickelt für höchste Performance.

Ein anderer Auslöser war sicher das Aufkommen von immer mehr mobilen Devices wie Smartphones und Tablets, aber auch die immer häufiger vorkommenden Micro-Computer in allen möglichen kleinsten Maschinen, bis hin zum Internet of Things (IOT). All diese Geräte haben eines gemeinsam: wenige Ressourcen. Programme für diese Geräte sollen also wenig Speicher verbrauchen, hohe Performance abliefern und dabei noch wenig Energie verbrauchen. Außerdem ist hier häufig Hardware-Nähe angesagt. Erinnern wir uns: All das zeichnet C aus und natürlich auch C++.

Aber natürlich gab es auch noch andere Gründe, die für C++ sprachen. Die Zeit war einfach wieder reif für C++. Universitäten und Firmen investierten wieder in C++ und das Standardisierungs-Komitee setzte die Weiterentwicklung von C++ intensiv fort.

Im September 2011 wurde der dritte ISO C+ Standard verabschiedet: „ISO/IEC 14882:2011" oder kurz C++11. C++11 war ein sehr großer Schritt nach vorne für die Programmiersprache. C++11 brachte nicht einfach nur ein paar kleine Erweiterungen mit sich, sondern einen großen Strauß an neuen Features. Für den Einsteiger wurde die Programmiersprache einfacher, für den normalen Entwickler eleganter und für den Experten viel ausdrucksstärker. Bjarne Stroustrup sagte damals: „C++11 fühlt sich an wie eine neue Sprache." Man sieht den großen Schritt auch im Umfang des Standards. Dieser war noch für C++03 insgesamt 768 Seiten lang, der C++11-Standard benötigte schon 1.472 Seiten.

3.2.3 C++14

Nach C++11 änderte das Standardisierungs-Komitee sein Vorgehen. Sowohl C++98 als auch C++11 hatten sich stark verspätet. Das Komitee hatte sich einen gewünschten Feature Umfang gesetzt – und der Standard wurde erst dann verabschiedet, als alle diese Features fertig waren. Damit konnte ein Feature, das Probleme macht, den gesamten Zeitplan umwerfen.

Das neue Vorgehen ist nicht mehr feature-basiert, sondern zeit-basiert. Das Ziel ist, alle drei Jahre einen neuen Standard zu veröffentlichen. Was fertig ist, ist drin. Was nicht fertig ist, muss bis zum nächsten Standard warten. Ist nur wenig fertig, dann ist der neue Standard nur eine kleine Änderung – ist viel fertig, dann ist er ein großer Schritt. Bislang funktioniert das neue Verfahren – es gab Standards in 2014, 2017, 2020 – und der nächste Standard ist schon fast auf der Zielgraden für 2023.

Im Dezember 2014 wurde C++14 publiziert („ISO/IEC 14882:2014"). C++14 war nur ein kleiner, aber wichtiger Schritt. Es gab nur wenige wichtige Neuigkeiten – wie zum Beispiel generalisierte Lambdas. Aber viele Themen aus C++11 wurden erweitert, abgerundet und verfeinert. Viele Fehler wurden behoben und eine Menge kleinerer Dinge wurde ergänzt, die man in C++11 zum Teil vergessen hatte.

3.2.4 C++17

Wieder im Dezember, aber noch rechtzeitig in 2017, wurde der bislang letzte und daher aktuelle Standard C++17 („ISO/IEC 14882:2017") veröffentlicht. Viele hatten sich für C++17 einen großen Wurf erhofft – vergleichbar zu C++11. Aber leider sind viele Features nicht rechtzeitig fertig geworden, und – wir erinnern uns – was nicht fertig ist, ist nicht drin. Gegenüber den großen Erwartungen war C++17 also wieder nur ein kleiner Schritt – ich würde ihn aber als mittleren Schritt bezeichnen. Es gab zwar nur wenige größere neue Features und Erweiterungen, aber in der Gesamtheit hat sich an vielen Stellen etwas geändert – und einige dieser Stellen sind schon wichtig.

3.2.5 C++20

2020 passierte dann wieder ein großer Schritt für die C++ Gemeinschaft. C++20 wurde verabschiedet. C++20 ist wieder ein großer Fortschritt in der C++ Programmierung, da der Standard neben sehr vielen kleinen und mittleren Verbesserungen mit Modulen, Concepts, Coroutinen, Ranges und dem Space-Ship Operator auch fünf große neue Features enthalten hat, die die Art, wie wir C++ programmieren, wieder verändert hat.

C++20 ist zurzeit der aktuelle Standard. Daher bezieht sich dieses Buch auf C++20, und erklärt und nutzt einige der neuen Features.

3.2.6 C++23

An C++23 wird aktuell gearbeitet. Auch wenn C++23 wohl erst in der zweiten Jahreshälfte von 2023 verabschiedet wird, so wissen wir heute schon sehr genau, was er enthalten wird. Und manche der kommenden Features sind auch schon von den aktuellen Compilern implementiert. Trotzdem ist der genaue Umfang von C++23 ein Blick in die Glaskugel. Es ist jetzt schon sicher, dass C++23 eher wieder ein kleinerer Schritt sein wird – er wird also wie C++14 für C++11 in erster Linie Fehler berichtigen, Lücken füllen und kleinere Erweiterungen einführen. Trotzdem wird auch C++23 einige sehr interessante Neuigkeiten, wie zum Beispiel „print“ oder „Deducing This“, enthalten. Aber das ist noch Zukunftsmusik.

3.3 Die Programmier-Paradigmen

C++ ist eine sogenannte Multi-Paradigmen-Sprache. Damit wird gemeint, dass C++ nicht nur ein Programmier-Paradigma unterstützt oder es zentral für die Programmiersprache ist, sondern dass die Sprache viele Paradigmen unterstützt. Bjarne Stroustrup hat mal gesagt, dass C++ nicht entwickelt wurde, um eine Idee zu beweisen, sondern um reale Probleme zu lösen. Und für reale Probleme benötigt man oft

viele Werkzeuge und Paradigmen – und C++ versucht, viel davon bereitzustellen.

Die beiden zentralen Programmier-Paradigmen von C++ sind die Objekt-Orientierung, die C++ mit Klassen, Vererbung, Polymorphie und anderen Elementen unterstützt, und das generative Paradigma, für das C++ Templates, generische Lambdas, auto, decltype und weitere Features anbietet. Aber C++ enthält zum Beispiel auch viele funktionale Elemente, die man zum Beispiel in den Lambdas, Algorithmen, Ranges und Funktions-Zeigern wiederfindet.

Zusätzlich zu den unterstützten Programmier-Paradigmen steht C++ auch stark in der Historie von C. C++ hat sich immer auch als ein besseres und moderneres C verstanden – auch wenn eingefleischte C Programmierer das anders sehen mögen. C++ ist halt aus C entstanden, und Bjarne Stroustrup wollte die guten Seiten von C auf jeden Fall in C++ behalten. Außerdem sollte C++ möglichst abwärtskompatibel zu C sein, daher möglichst viele C Programme sollten problemlos als C++ Programm compilierbar sein. Diese Philosophien haben sich bis heute erhalten. C++ steht weiterhin in der Tradition von C, und Erweiterungen in C++ sollen möglichst wenig alten Code brechen. Mittlerweile haben sich einige Änderungen eingeschlichen, die aber normale C Programme nicht betreffen sollten. So werden zum Beispiel seit C++17 keine sogenannten Trigraph-Sequenzen mehr unterstützt – dies sind spezielle Zeichenkombinationen, die auf den Terminals von alten Großrechnern notwendig waren, da diese zum Beispiel keine spitzen Klammern „<“ und „>“ unterstützten.

Die wichtigsten Sprach-Paradigmen von C sind „Kontrolle“ und „Effizienz“ – und die hat C++ ohne Einschränkung von C übernommen. Kontrolle meint, dass man als Programmierer volle Kontrolle über die Effekte seines Programms hat. So kann man in C++ zum Beispiel das Speicherlayout von Objekten genau definieren – daher können Sie als Programmierer genau festlegen, wie Ihre Variablen im Speicher aussehen und wo sie liegen. Oder C++ fügt von alleine keine versteckten Elemente zu Ihrem Code hinzu, die Speicher und/oder Performance kosten. Es gibt zwar Sprachfeatures, die etwas „kosten“ – aber die müssen

Sie explizit anfordern. In C++ gilt die Philosophie: Sie bezahlen (in Form von Speicher und/oder Performance) nur für Dinge, die Sie explizit verlangen. Sie haben volle Kontrolle über die Effekte Ihres Programms.

Wer meint, dass diese Kontrolle nicht notwendig ist – und fast alle anderen Programmiersprachen bieten sie auch nicht an –, muss daran denken, dass C und C++ zum Beispiel auch für das Schreiben von Embedded-Systemen, Hardware-Treibern oder Betriebssystemen gedacht sind. Und wenn Sie zum Beispiel im Sensor einer Wetterstation nur 3 KByte Speicher und einen schwachen Prozessor zur Verfügung haben (und das ist kein Tippfehler – ich meine wirklich Kilo-Byte), dann müssen Sie auf diese Dinge Wert legen. Aber auch bei Apps, Desktop- oder Server-Programmen ist die hohe Effizienz immer hilfreich, und oft auch zwingend erforderlich.

Auf der anderen Seite hat C++ viele abstrakte Konzepte in die Programmiersprache aufgenommen, um die „einfache“ Implementierung großer Projekte zu ermöglichen, um zum Beispiel verteilte Systeme umzusetzen und vieles mehr. Aber keiner dieser abstrakten Konzepte durfte unnötigen Overhead in die Laufzeit einbringen – Sie bezahlen nur für das, was Sie anfordern.

Dieser Spagat ist auf der einen Seite die Stärke von C++, aber gleichzeitig auch die Schwäche. Durch die vielen Möglichkeiten kann man in C++ sehr kleine und effiziente Programme schreiben, die genau das machen, was man möchte. Aber um diese Kontrolle anzubieten, enthält die Sprache viele Details, die C++ recht komplex machen. Um zum Beispiel eine Variable an eine Funktion zu übergeben – eine ganz alltägliche Aufgabe beim Programmieren –, bieten die meisten Programmiersprachen nur eine Art an. Ganz einfach und trivial. In C++ gibt es hierfür über 10 verschiedene Möglichkeiten, die sich zum Teil nur minimal unterscheiden. Man muss hier beim Programmieren also nachdenken und sich entscheiden – dafür hat man die volle Kontrolle. Ich vergleiche dies gerne mit dem einfachen Bastler, der einfach einen simplen Schwingschleifer im Haus hat und diesen für alles nutzt. Der versierte Heimwerker aber hat neben dem Schwingschleifer noch einen Dreiecksschleifer, einen Extenderschleifer, eine Bandschleifmaschine,

und sicher jede Menge Feilen und Raspeln, und auch das ganz normale alte Schmirgelpapier zuhause. Für jeden Anwendungsfall das passende Werkzeug. Und er kann damit wahre Wunder vollbringen. Wenn Sie das auch wollen – willkommen in der Wunderwelt von C++.

Daher ist C++ auch keine gute Programmiersprache für den Gelegenheitsprogrammierer. Wenn Sie nur einmal im Jahr ein paar Zeilen programmieren, um zum Beispiel irgendeine Aufgabe zu automatisieren – dann ist C++ nicht die richtige Sprache für Sie. Mit C++ muss man sich beschäftigen. Dafür entschädigt einen die Sprache dann mit sehr mächtigen und eleganten Ideen. Aber keine Panik – die Situation ist viel besser geworden. C++11, auch C++14 und C++17, und vor allem C++20 enthalten extra deshalb viele neue Features, die die Programmiersprache auch viel einfacher gemacht haben.

Kapitel 4

Praxis

In diesem Kapitel wollen wir uns die praktische Seite der C++ Programmierung anschauen und dabei lernen, welche Werkzeuge wir zum C++ Programmieren benötigen. Richtig starten mit dem Programmieren werden wir erst im nächsten Kapitel.

4.1 Programmier-Ablauf

Bevor wir mit dem eigentlichen Programmieren und dem Lernen von C++ beginnen, wollen wir uns anschauen, wie ein typischer Programmier-Ablauf in C++ aussieht.

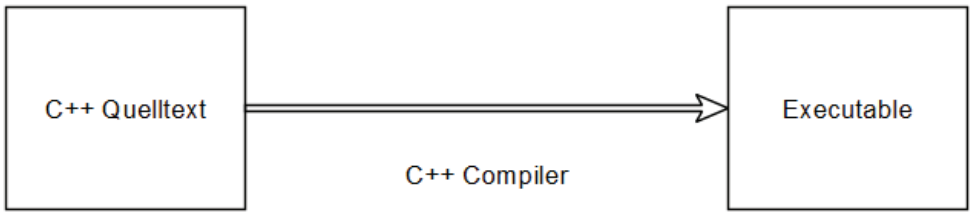


Abb. 4-1 Programmier-Ablauf

Was wir machen, wenn wir C++ programmieren, ist C++ Code zu schreiben. Dieser Code kann nicht direkt von einem Computer ausgeführt werden, sondern muss vorher übersetzt werden. Diesen Vorgang nennt man compilieren, und das Tool, das wir dazu nutzen, ist ein sogenannter Compiler. Der Compiler übersetzt unseren Quelltext und erzeugt ein ablauffähiges Programm – das sogenannte Executable.

4.1.1 Code beziehungsweise Quelltext

Wenn wir mal von grafischen Programmiersprachen absehen, von denen es nur wenige gibt, und die auch fast ausschließlich für Lehrzwecke oder spezielle Anwendungen verwendet werden, besteht Programmie-

ren in allen Programmiersprachen daraus, seine Ideen in Code auszudrücken und als Text aufzuschreiben. Dieser Text wird oft Quelltext genannt, da er die Quelle für das ausführbare Programm ist.

Um Code (oder Quelltext) zu schreiben, benötigt man ein Programm – einen sogenannten Editor (oder auch Text-Editor) genannt. Sie können Code nicht mit einer Text-Verarbeitung wie „Open-Office", „In-Design" oder „MS-Word" schreiben. Code ist einfach nur reiner Text – ohne Formatierungen, Tabellen, Überschriften und vielem mehr. Und darum funktionieren Text-Verarbeitungen nicht als Werkzeug, um Code zu schreiben. Die Anforderungen sind komplett diametral.

Benutzen Sie für die Entwicklung einen Editor oder alternativ eine integrierte Entwicklungs-Umgebung (IDE). Im nächsten Abschnitt werden wir Näheres dazu erfahren.

4.1.2 C++ Compiler

Der C++ Code muss jetzt in ein ausführbares Programm übersetzt werden. Dies ist die Aufgabe eines C++ Compilers. Genau genommen ist dies ein dreistufiger Prozess mit drei involvierten Tools, dem Präprozessor, dem Compiler und dem Linker – aber diese Details lernen wir erst im zweiten Buch kennen. Man redet hier auch gerne vom Triumvirat aus Präprozessor, Compiler und Linker. In den meisten Fällen reden wir einfach vom Compiler und vom Compilieren – auch wenn sich dahinter mehr verbirgt. Nur wenn es wirklich genau um eine der Phasen geht, dann drücken wir uns exakt aus. Ansonsten redet man landläufig immer nur vom Compilieren.

Hinweis: Andere Programmiersprachen arbeiten hier anders. Es gibt neben dem Compiler als Sprach-Werkzeug zum Beispiel auch sogenannte Interpreter, aber auch Mischformen. Und auch im C++ Ökosystem findet man zum Beispiel mit dem „Cling" einen C++ Interpreter. Aber die normale Art, in C++ zu programmieren, ist die Nutzung eines C++ Compilers. Darauf beschränken wir uns hier.

Compiler, Interpreter & Mischformen

Ein Compiler übersetzt das Programm in ein vom Computer direkt ausführbares Programm, das sogenannte Executable (vergleiche Abb. 2.1). Ein Interpreter dagegen nimmt den Quelltext und führt diesen direkt aus. In diesem Fall muss auf dem Ausführungs-Computer dieser Interpreter installiert sein, da der Computer von sich aus den Quelltext nicht ausführen kann. Typische Interpreter-Sprachen sind zum Beispiel Ruby oder Python.

Eine andere Variante ist die Nutzung einer virtuellen Maschine. Hierbei wird der Quelltext von einem Compiler in sogenannten Byte-Code übersetzt. Den kann der Computer von sich aus nicht ausführen. Dazu wird dann eine virtuelle Maschine benötigt – eine Art Interpreter für den Byte-Code. Eine typische Compiler-Sprache mit virtueller Maschine ist Java.

Alle Vorgehensweisen haben ihre spezifischen Vor- und Nachteile. C++ ist von der Grund-Philosophie her eine Compiler-Sprache. Andere Vorgehensweisen betrachten wir im Buch daher nicht näher.

Mehr über C++ Compiler erfahren Sie im übernächsten Abschnitt.

4.1.3 Dokumentation

Auch wenn Sie dieses C++ Buch in Händen halten und es nicht dünn ist – es steht bei weitem nicht alles hier drin. Darum wird es bald noch ein zweites Buch geben, aber auch damit werden Themen offenbleiben. Würde ich zum Beispiel jede Funktion der C++ Standard-Bibliothek aufführen und erklären, dann bräuchten wir mindestens 4 Bücher. Und auch die Sprache C++ enthält noch viele Details und Anwendungen, die den Umfang dieser beiden Bücher sprengen. Und es wird im Laufe des Lesens sicher eine oder andere Frage von Ihnen auftauchen, die diese Bücher nicht oder nur knapp beantwortet – irgendwas fehlt immer.

Also seien Sie froh, dass das Buch nicht dicker ist. Aber trotzdem werden Sie auf Dauer auf viele dieser Informationen angewiesen sein oder eine Anlaufstelle für Ihre Fragen benötigen. Ein paar Hinweise finden Sie im Abschnitt nach den C++ Compilern.

4.1.4 Bibliotheken

Reales Programmieren heißt oft auch pragmatisch handeln – sonst werden Sie nie fertig mit Ihren Projekten. Und zum pragmatisch handeln gehört auch, dass Sie nicht alles selbst programmieren, sondern sich auf die Arbeit anderer Programmierer abstützen – zumeist in Form von fertigen C++ Bibliotheken und Frameworks. Darum gibt es danach auch einen kurzen Abschnitt über Bibliotheken. Was genau eine Bibliothek ist, wie man sie einbindet, wie man sie selber schreibt und viele andere Dinge mehr – das besprechen wir später im Buch noch detaillierter.

4.1.5 Fazit

Um also in C++ zu programmieren, benötigen Sie mindestens:

- Einen Editor oder eine IDE (integrierte Entwicklungs-Umgebung)
- Einen C++ Compiler
- Referenz-Dokumentation
- Später sicher noch weitere C++ Bibliotheken

4.2 Editoren & Entwicklungs-Umgebungen

4.2.1 Editoren

Wie wir gelernt haben, benötigen wir zum Schreiben von Code einen Editor. Aber welchen soll ich benutzen? Und woher bekomme ich ihn?

Nun, jedes Betriebssystem bringt schon einen Editor mit – von daher bräuchte man sich eigentlich nicht um dieses Thema zu kümmern.

Aber nicht jeder Editor ist wirklich gut zum Programmieren geeignet. Man hat hier aber die riesige Auswahl, denn zum Glück gibt es Editoren wie Sand am Meer. Nicht umsonst sagt man, dass irgendwann mal jeder ambitionierte Programmierer seinen eigenen Editor geschrieben hat.

Der unter Windows mitgelieferte Editor „Notepad“ (oder genau eigentlich „MS-Editor“ – „https://de.wikipedia.org/wiki/Microsoft_Editor“) ist extrem einfach und zum Programmieren nicht wirklich zu gebrauchen. Zum Glück gibt es viele freie und käufliche Alternativen. Hier ein Vorschlag für zwei recht verbreitete freie Editoren unter Windows. Aber Achtung: Wenn Sie mit anderen Programmierern reden, dann kann es Ihnen passieren, dass Sie diesen Editoren furchtbar finden und nicht verstehen, warum Sie nicht Editor „xyz“ benutzen. Editoren sind unter Programmierern ein fast schon religiöses Thema mit manchmal entsprechendem Fanatismus. Darum – dies sind nur Vorschläge – auch andere Editoren sind sehr gut.

4.2.2 Notepad++

- Ein guter freier Editor, der sehr viele hilfreiche Features zum Programmieren mitbringt und sich dabei immer noch relativ einfach bedienen lässt. Er ist nur für Windows verfügbar.
- http://notepad-plus-plus.org/
- https://de.wikipedia.org/wiki/Notepad%2B%2B

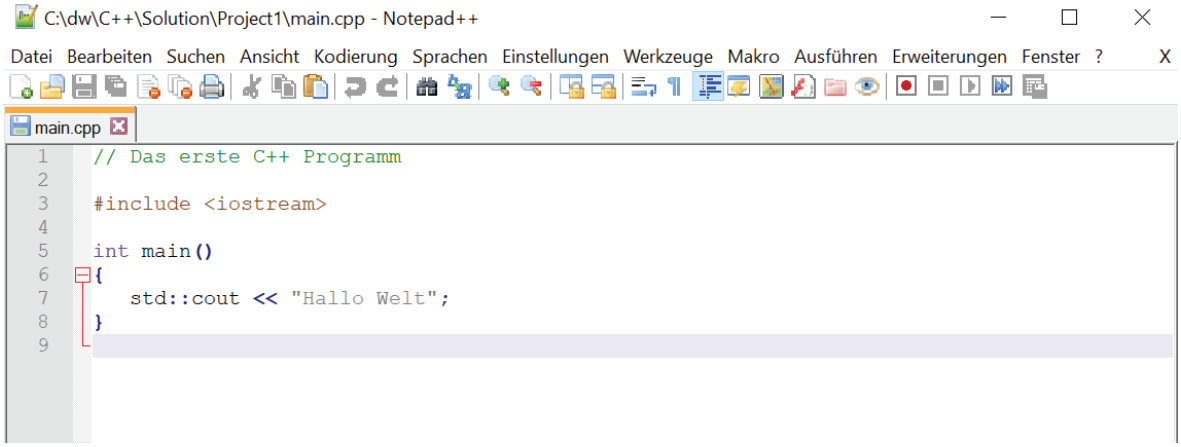


Abb. 4-2 Notepad++ mit C++ „Hallo Welt“ Programm

4.2.3 Microsoft Visual Code

- Ein extrem leistungsfähiger freier Editor von Microsoft, der unter Windows, Mac OS-X und Linux läuft. Nicht zu verwechseln mit der IDE Microsoft Visual Studio – trotz des sehr ähnlichen Namens.
- https://code.visualstudio.com/
- https://de.wikipedia.org/wiki/Visual_Studio_Code

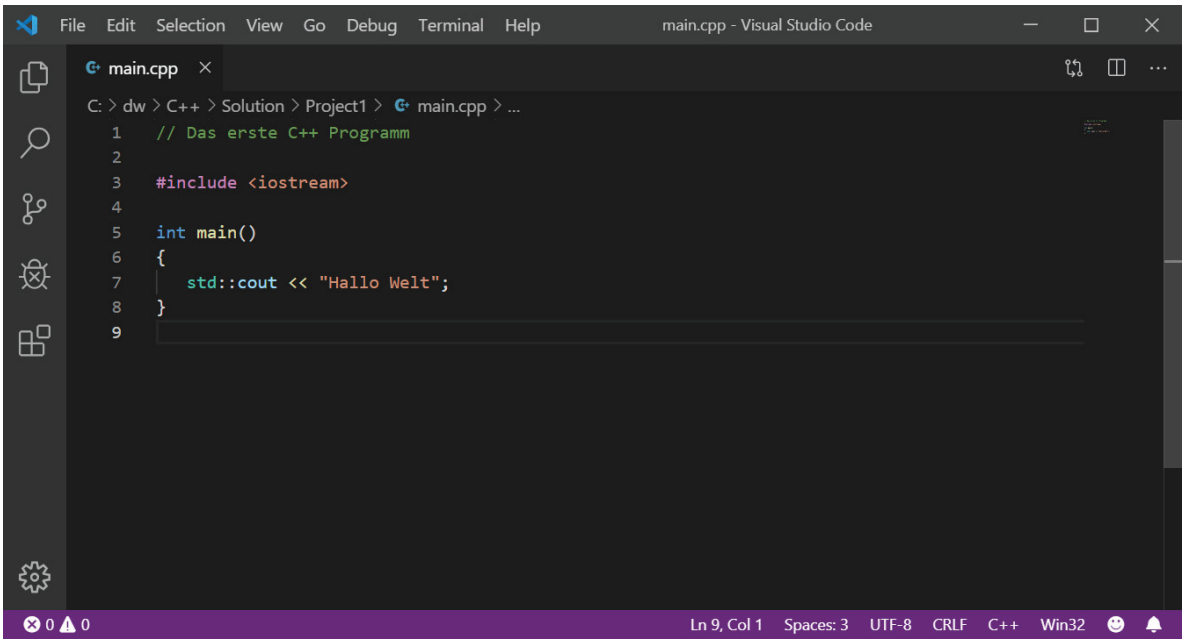


Abb. 4-3 Microsoft Visual Code mit C++ „Hallo Welt“ Programm

Unter Linux sind die typischen Editoren „vi“ beziehungsweise „vim“ und „Emacs“, die beide im Systemumfang enthalten sind. Beide sind sehr leistungsfähig, aber auch recht gewöhnungsbedürftig in ihrer Bedienung, so dass sich viele normale Benutzer am Anfang sehr schwertun. Wenn Sie unter Unix arbeiten wollen, kann ich Ihnen aber trotzdem nur empfehlen, zumindest einen der beiden etwas zu beherrschen.

- https://de.wikipedia.org/wiki/Vi
- https://de.wikipedia.org/wiki/Vim
- https://de.wikipedia.org/wiki/Emacs

4.2.4 IDE's

Komfortabler als mit einem Editor empfinden die meisten Programmierer das Arbeiten in einer IDE („Integrated Development Environment"), daher in einem Programm, das Editor, Compiler und meist noch viele andere Hilfsmittel unter einer – meist grafischen – Oberfläche zusammenfasst (integriert).

Die Vorteile einer IDE sind eine einheitliche Oberfläche, viele hilfreiche Tools und eine gute Integration zwischen den Tools (so führt zum Beispiel in den meisten IDE's ein Doppelklick auf eine Fehlermeldung direkt in den Editor zu der Zeile mit dem Fehler). Außerdem benötigt man am Anfang nur ein Programm, das alles Notwendige enthält.

Es gibt eine Unmenge an Programmen, die im weitesten Sinne als IDE aufgefasst werden können, da sie zum Beispiel vorhandene Tools unter einem Dach integrieren und/oder sie durch eigene Tools erweitern oder ersetzen. Die deutsche Wikipedia bietet einen guten Einstieg für die Suche: „https://de.wikipedia.org/wiki/Liste_von_Integrierten_Entwicklungsumgebungen".

Hinweis: Auch in den Zeiten von IDE's haben Editoren, Kommandozeilen-Compiler und andere Tools noch ihre Berechtigung. In der Praxis werden beide Systeme oft neben einander eingesetzt. Während der einzelne Entwickler (meist) in einer IDE entwickelt, werden die normalen Tools zum Beispiel für die Cross-Plattform Compilierung, Nightly-Builds oder automatisierte Deployments genutzt.

4.2.5 Microsoft Visual Studio Community 2022

Unter Windows ist die verbreitetste IDE für die C++ Entwicklung wohl das Microsoft Visual Studio, das direkt auch den C++ Compiler von Microsoft enthält. Aber auch andere IDE's, wie zum Beispiel die Eclipse CDT, CodeBlocks oder Qt-Creator, sind weit verbreitet.

Wenn in diesem Buch auf die praktische Nutzung einer IDE eingegangen wird, wird dies das Microsoft Visual Studio Community 2022 für

C++ sein. Das soll nicht bedeuten, dass die anderen IDE's schlecht sind. Ganz im Gegenteil: Sie haben in manchen Bereichen Vorteile gegenüber dem Microsoft Visual Studio, in anderen Nachteile. Es ist auch viel Geschmackssache. Die Community Edition des Microsoft Visual Studios 2022 können Sie als einzelner Entwickler kostenlos herunterladen und nutzen: „https://visualstudio.microsoft.com/de/vs/community/". Achten Sie bei der Installation darauf, dass Sie auch die C++ Unterstützung mit installieren. Am Ende dieses Kapitels finden Sie daher eine kurze Anleitung für die Installation.

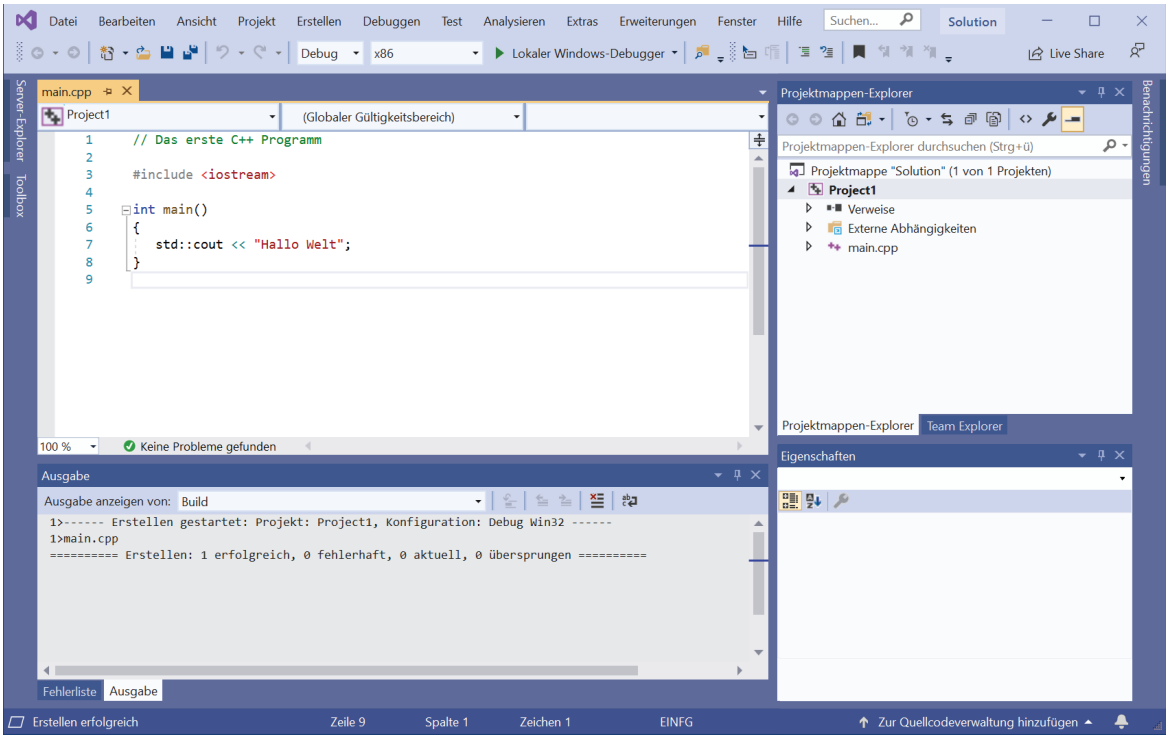


Abb. 4-4 Microsoft Visual Studio Community mit C++ „Hallo Welt" Programm

Im nächsten Kapitel werde ich in die erste Nutzung des Microsoft Visual Studios einführen.

4.3 C++ Compiler

Wenn Sie mit dem Microsoft Visual Studio arbeiten, dann haben Sie automatisch einen C++ Compiler mit installiert, und Sie können quasi

direkt loslegen. Die meisten IDE's bringen aber keinen C++ Compiler mit, sondern benötigen einen fertig installierten C++ Compiler.

Es gibt vier C++ Compiler, die am häufigsten eingesetzt werden – und dadurch für die C++ Community am wichtigsten sind.

Visual C++

- Der Visual C++ Compiler von Microsoft ist der im Microsoft Visual Studio integrierte C++ Compiler. Er ist der unter Windows verbreitetste Compiler und ist ein kommerzielles Produkt, aber in der Community Edition für Studenten und einzelne Entwickler auch frei erhältlich. Er kann auch im Zusammenspiel mit anderen IDE's als dem Microsoft Visual Studio genutzt werden – die beste Integration findet man aber dort.
- https://de.wikipedia.org/wiki/Visual_C%2B%2B

G++ (GCC)

- Der g++ ist der C++ Compiler aus der GNU Compiler Collection (GCC) und wird auch oft als GCC bezeichnet. Er ist wohl der am häufigsten eingesetzte C++ Compiler unter Linux, ist aber auch unter Windows (als MinGW Port) und unter Mac OS-X vorhanden. Er ist ein Open-Source Compiler und unterliegt der GNU General Public License.
- https://de.wikipedia.org/wiki/GNU_Compiler_Collection
- https://de.wikipedia.org/wiki/MinGW

Clang

- Der Clang ist das Compiler-Frontend für C++ aus dem LLVM Projekt. Auch er ist – analog zum G++ – ein Open-Source Compiler und unter Linux, Windows und Mac OS-X verfügbar. Er unterliegt einer BSD-ähnlichen Lizenz. Das Besondere am Clang und dem LLVM Projekt ist, dass er intern sehr modular und sauber aufgebaut ist und sich dadurch gut in anderen Kontexten einsetzen lässt. So gibt es mittlerweile viele Werkzeuge für C++, die auf die Clang und LLVM aufsetzen.

Seit dem Erscheinen des Clang 2007 ist das C++ Ökosystem stark gewachsen.

- https://de.wikipedia.org/wiki/Clang
- https://de.wikipedia.org/wiki/LLVM

Intel C++ (ICC)

- Der Intel C++ Compiler (ICC) ist ein kommerzieller Compiler der Firma Intel für die Betriebssysteme Linux, Windows und Mac OS-X. Da er kommerziell ist, ist er nicht so verbreitet wie die anderen drei Compiler. Er hatte den Ruf, den effizientesten Maschinen-Code für Intel Prozessoren zu erzeugen. Da die anderen Compiler aber ständig verbessert worden sind, weiß ich nicht, ob dieser Ruf heute noch gerechtfertigt ist.
- https://de.wikipedia.org/wiki/Intel_C%2B%2B_Compiler
- https://software.intel.com/en-us/compilers

Einen Überblick über viele C++ Compiler finden Sie zum Beispiel hier:

- https://de.wikipedia.org/wiki/C%2B%2B#C++-Compiler
- https://isocpp.org/get-started

4.3.1 ISO C++ Konformität

Ein großes Thema bei C++ Compilern ist die ISO C++ Konformität. Daher welche Sprach- und Bibliotheks-Features unterstützen die Compiler. Damit ist das Thema gemeint, welche C++ Sprach- und Biblotheks-Features der jeweilige Compiler unterstützt. In C++ sind die Veröffentlichungs-Zyklen der ISO C++ Standards und die Zyklen der C++ Compiler voneinander entkoppelt. Das Standardisierungs-Gremium veröffentlicht alle drei Jahre einen neuen C++ Standard. Und die Compiler-Bauer veröffentlichen davon unabhängig in ihren eigenen Intervallen neue C++ Compiler Versionen. Dies hat zur Folge, dass manche Compiler Versionen schon Features unterstützen, die erst in kommenden Standards

vielleicht kommen werden – oder aber auch, dass noch nicht alle Features des aktuellen oder letzten Standards unterstützt werden.

Wenn Sie mit mehreren Compilern entwickeln, zum Beispiel weil Sie auf mehreren Plattformen unterwegs sind, dann müssen Sie hier gut aufpassen. Sie können nur die Features einsetzen, die alle Ihre Compiler unterstützen. Wenn Sie in einem größeren Projekt arbeiten – sei es beruflich oder privat –, dann kann es Ihnen passieren, dass – aus was für Gründen auch immer – ältere Compiler eingesetzt werden. Und dann müssen Sie sich auf die Features beschränken, die Ihnen dort zur Verfügung stehen. Einen Überblick über die unterstützten C++ Features in den verschiedenen Compilern und Versionen finden Sie zum Beispiel hier: „https://en.cppreference.com/w/cpp/compiler_support".

Im Prinzip kann ich Ihnen nur empfehlen: Wenn Sie irgendwie können, dann nutzen Sie einen möglichst modernen Compiler, da die neuen Features das Programmierer-Leben oft stark vereinfachen und erleichtern. Aber auch darüber hinaus haben neuere Compiler oft viele neue beziehungsweise verbesserte hilfreiche Funktionalitäten.

Wir schauen uns in diesem Buch C++20 an. Dies ist der aktuelle Standard. Wenn Sie also Beispiele aus diesem Buch mit Ihrem Compiler nicht compiliert bekommen, so kann dies an zwei Dingen liegen:

- Sie verwenden einen älteren Compiler, der ein benutztes Sprach- oder Bibliotheks-Feature noch nicht unterstützt. Da bleibt Ihnen nichts anderes übrig, als auf einen neueren Compiler zu wechseln, oder dieses Feature nicht zu nutzen.
- Ein anderes Problem kann sein, dass Sie Ihren Compiler für C++20 konfigurieren müssen. Alle Compiler compilieren mit einer Default-Einstellung, und die ist bei den meisten Compilern zurzeit C++11 oder C++14. Achtung: So etwas ändert sich von Version zu Version – diese Aussage kann also sehr schnell überholt sein. Und dann müssen Sie dem Compiler sagen, welche ISO C++ Version Sie nutzen wollen. Für den Microsoft Visual C++ Compiler besprechen wir das im nächsten Kapitel in der Einführung des Microsoft Visual Studios. Für den G++

und den Clang ist dies für C++20 zum Beispiel die Kommandozeilenoption „-std=c++20“.

4.3.2 Optimierungen

Eine wichtige Aufgabe von C++ Compilern ist die Performance- und Speicher-Optimierung. Der C++ Compiler darf und soll den C++ Code so übersetzen, dass der entstandene Maschinen-Code möglichst wenig Speicher benötigt und möglichst performant auf der Ziel-Plattform läuft. Es dreht sich in C++ nicht alles – aber halt sehr viel – um Performance. Und neben der Nutzung effizienter Algorithmen und dem Schreiben von gutem C++ Code ist die dritte Möglichkeit, Performance zu gewinnen, ein Compiler, der aggressiv optimiert. Er darf dabei das Verhalten des Programms nicht verändern – also das, was der Programmierer geschrieben hat, muss semantisch passieren. Aber wie es letztlich beim Ausführen passiert, ist egal: Hauptsache, es ist performant.

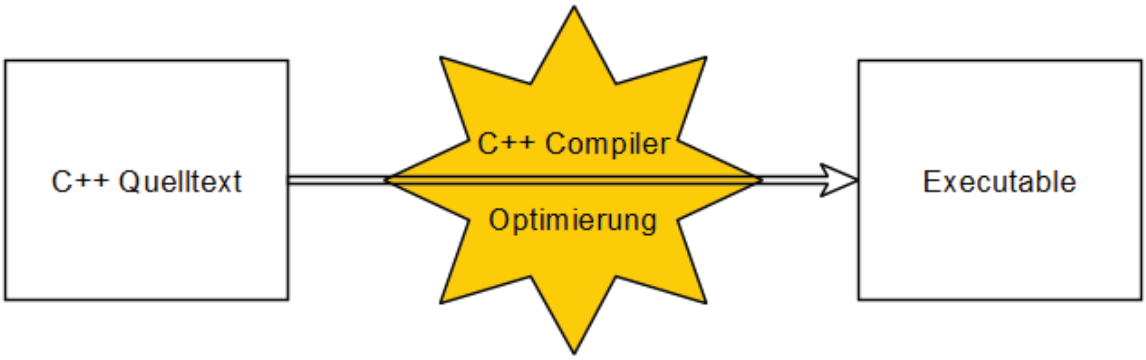


Abb. 4-5 Optimierung

Schauen wir uns ein paar Beispiele an. Wenn Sie sie noch nicht verstehen, ist das nicht schlimm. Kommen Sie später noch mal kurz hierher zurück.

```
1  int x = 3 + 4;
```

Hier wünschen wir, dass der Compiler erkennt, dass „3+4“ immer „7“ ergibt, und der Compiler diesen Code intern zu diesem umbaut:

```
1  int x = 7;
```

Oder der Compiler könnte Schleifen ausrollen. Dieser Code

```
1 for (int i=0; i<3; ++i)
2 {
3    fct(i);
4 }
```

würde zu diesem werden:

```
1 fct(0);
2 fct(1);
3 fct(2);
```

Schon sehr viel anspruchsvoller wäre folgende Optimierung, die heutige Compiler aber beherrschen:

```
1 int res = 0;
2 for (int i=1; i<=3; ++i)
3 {
4    res += i;
5 }
```

Gute Compiler machen daraus:

```
1 int res = 6;
```

Solche und viele weitere Optimierungen beherrschen heutige C++ Compiler.

Unser primäres Ziel als C++ Programmierer wird es sein, Code zu schreiben, der fehlerfrei ist und gut zu lesen ist. Aber sekundär soll unser Code den Optimierungen des Compilers keine Steine in den Weg legen, sondern ganz im Gegenteil ihm zu ermöglichen, viele Optimierungen durchzuführen. Immer wieder im Buch werden wir die Auswirkungen dieser C++ Philosophie sehen. Aber dazu später mehr. Merken Sie sich hier, dass Performance in C++ wichtig ist und wir die Compiler-Optimierungen unterstützen wollen.

4.4 Bibliotheken

Eine Programmiersprache definiert die grundlegenden Konstrukte, um Computer-Programme zu schreiben. Wenn man ein Programm schreibt, dann werden viele Probleme immer wieder anfallen – zum Beispiel das Verarbeiten von Texten. Um solche Probleme nicht immer wieder neu lösen zu müssen, verwendet man beim Programmieren fertige Bibliotheken, die Lösungen für solche und andere Probleme enthalten. Natürlich kann man auch eigene Bibliotheken schreiben – wir werden das später besprechen.

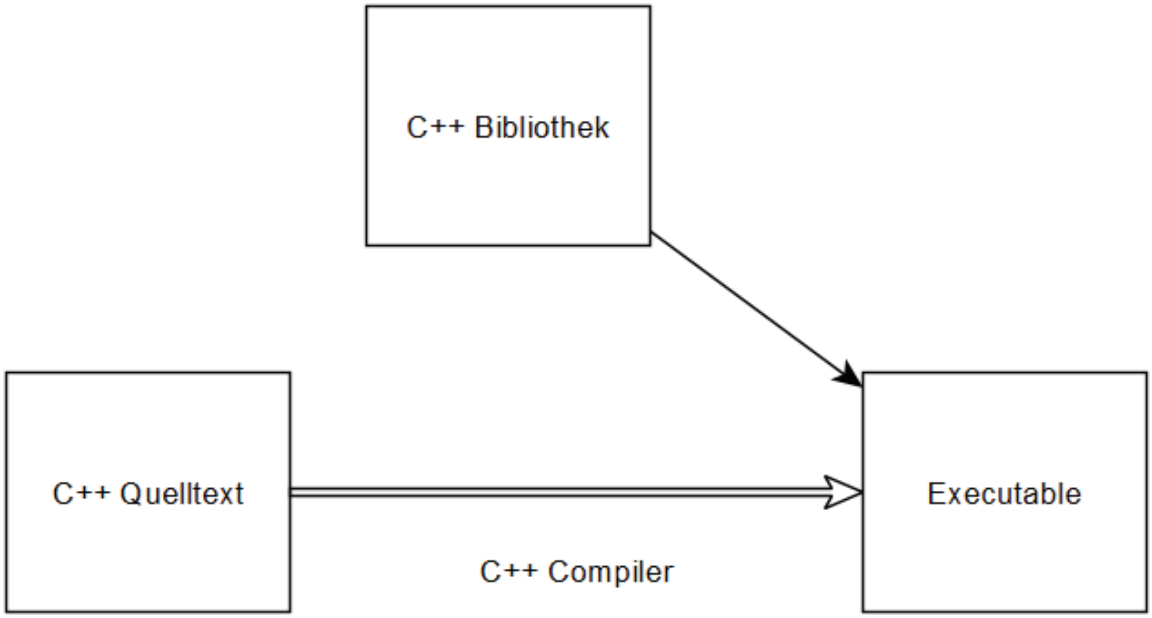


Abb. 4-6 Compilierung mit Bibliothek

Neben der eigentlichen Sprache C++ definiert der ISO Standard auch eine Bibliothek, die viele grundlegende Elemente und Bausteine enthält, die man zum Programmieren in C++ benötigt. Diese Standard-Bibliothek wird nur zu einem kleinen Teil in diesem Buch besprochen, da der Umfang zu groß ist und das Buch dann mehr eine Referenz wäre als ein Lehrbuch. Trotzdem benötigt man natürlich diese Informationen – im übernächsten Abschnitt finden Sie Links auf entsprechende Referenzen im Internet.

Wenn Sie in C++ ein Problem lösen müssen, dann sollten Sie zuerst schauen, ob es eine passende Lösung in der C++ Standard-Bibliothek

gibt. Es gibt eigentlich keinen guten Grund, etwas selbst zu implementieren, wenn es schon fertige Lösungen gibt. Ein grober Überblick über den Inhalt der Standard-Bibliothek ist also sehr hilfreich, und Sie sollten ihn sich daher am Ende des Buches, wenn Sie mehr wissen, ruhig verschaffen.

Aber die C++ Standard-Bibliothek ist nicht riesig, geschweige denn allumfassend. Es fehlen viele Themen darin. Zum Beispiel können Sie allein mit ISO C++ keine grafischen Oberflächen oder Spiele programmieren. Auch die Unterstützung von zum Beispiel Netzwerken oder Datenbanken vermissen viele Programmierer in C++ schmerzlich. Mit jedem neuen Standard kommen natürlich neue Elemente in die Standard-Bibliothek, aber das Standardisierungs-Gremium hat nur begrenzte Ressourcen und muss sich daher auf die wichtigsten Kern-Themen konzentrieren. Das sind die Teile, die immer benötigt werden und sehr elementaren Charakter haben.

Aber das ist nicht schlimm. Natürlich kann man auch in C++ Netzwerke oder Datenbanken ansprechen, oder zum Beispiel grafische Oberflächen oder Spiele programmieren. Entweder – nicht empfehlenswert – programmiert man mit Hilfe des Betriebssystems alle diese Dinge selber. Oder – viel besser – man benutzt fertige C++ Bibliotheken, in denen diese Dinge schon umgesetzt sind.

In der Praxis wird man größere Programme eigentlich nie ohne Fremd-Bibliotheken erstellen können, da man nicht alles selber neu erfinden und programmieren will. Wir werden im Laufe des zweiten Buches auch mehrere Fremd-Bibliotheken nutzen und in unsere Programme einbinden.

Es gibt da draußen Millionen von C++ Bibliotheken, für alles und jeden. Ich kann Ihnen also keinen ernsthaften Überblick anbieten. Aber auf eine spezielle Bibliothek und ein paar Bibliotheks-Übersichten möchte ich Sie hinweisen.

Die wohl wichtigste C++ Bibliothek ist „Boost“ (siehe nächster Abschnitt). Wenn Sie ein Problem haben und in der C++ Standard-Biblio-

thek nicht fündig werden, dann sollten Sie zuerst bei Boost nach einer fertigen Lösung schauen.

Wenn auch Boost keine Lösung anbieten kann, dann müssen Sie im Internet auf die Suche nach einer passenden C++ Bibliothek gehen. Die folgenden Web-Seiten bieten Ihnen einen ersten guten Einstieg, um für viele Themen einen Überblick an möglichen Kandidaten zu finden:

- https://cpp.zeef.com/faraz.fallahi
- https://cpp.libhunt.com/
- https://florianjw.de/en/good_libraries.html
- https://github.com/fffaraz/awesome-cpp
- https://de.wikipedia.org/wiki/Liste_von_GUI-Bibliotheken#C.2B.2B
- https://en.wikipedia.org/wiki/List_of_widget_toolkits
- https://www.reddit.com/r/cpp/comments/babfl5/a_pretty_big_list_of_c_gui_libraries

4.5 Boost

Boost ist eine Sammlung „freier, portabler, begutachteter“ C++ Bibliotheken. Der Zweck von Boost ist die Stärken von C++ durch qualitativ hochwertige und in die C++ Philosophie passende Bibliotheken zu erweitern, ohne dass diese gleich einen ISO Standardisierungs-Prozess durchlaufen müssen.

Viele Leute sagen: „Boost ist die Spielwiese der C++ Standardisierer.“ Das ist nicht ganz falsch, denn:

- Boost wurde von Mitgliedern des C++ Standardisierungs-Komitees gegründet.
- Viele Mitglieder des C++ Standardisierungs-Komitees sind aktiv im Boost-Umfeld tätig.

- Viele Boost-Bibliotheken sind mittlerweile Bestandteil des ISO C++ Standards geworden – gerade in C++11 sind viele Bibliotheken in den Standard übernommen worden. Und wahrscheinlich werden weitere Bibliotheken von Boost in die nächsten C++ Standards Einzug finden.
- Mit Boost besitzen die „C++ Standardisierer" eine Möglichkeit, neue Bibliotheken zu testen und ausreifen zu lassen, bevor sie in den Standard übernommen werden.

Schauen wir uns Boost einmal etwas detaillierter an:

- Homepage: http://www.boost.org
- Die aktuelle Version ist 1.82.0 (vom 14.4.2023).
- Es erscheint ca. alle 3 Monate eine neue Version von Boost.
- Boost umfasst mehr als 120 C++ Bibliotheken – von sehr kleinen bis zu sehr großen.
- Es sind ständig weitere Bibliotheken in Arbeit.
- Es gibt eine sehr großzügige Boost Software Lizenz (BSL), die keine Einschränkung bzgl. der Verwendung der Bibliotheken sowohl in kommerziellen als auch freien Programmen enthält.
- Viele Unix Distributionen, wie zum Beispiel Fedora, Debian oder NetBSD, enthalten die Boost-Bibliotheken als normalen Bestand-Teil der Distribution.
- Die Boost-Bibliotheken haben unter anderem aufgrund ihrer Nähe zur C++ Standardisierung und einer großen Community eine sehr hohe Qualität. Dies betrifft sowohl die äußere als auch die innere Qualität. Die äußere Qualität meint hier die Themen API, Schnittstellen, Konventionen, Dokumentation, vielfältige, offene und erweiterbare Einsetzbarkeit, Design und so. Während die innere Qualität die Implementierung meint.
- Die meisten Boost-Bibliotheken sind reines ISO C++, und sollten von jedem standard-konformen Compiler problemlos übersetzt und benutzt werden können. Die restlichen Boost-Bibliotheken kapseln

in einem ISO C++ Wrapper plattform-spezifische API's (Application Programming Interface), zum Beispiel die „Boost Filesystem-Library". Ihre Implementierung ist daher natürlich plattform-spezifisch. Daher liegen sie leider nicht für jede Compiler- und Plattform-Kombination vor.

- Jeder kann neue Bibliotheken einreichen – vor ihrer Annahme durchlaufen die Bibliotheken aber einen Review-Prozess, der auch in einer Ablehnung enden kann. Wird eine Bibliothek angenommen, so muss sie typischerweise vor der endgültigen Integration in Boost noch überarbeitet werden – der Review-Prozess ergibt meist eine Menge von Anmerkungen, Problem-Hinweisen, Verbesserungs-Vorschlägen und Wünschen hervor, die vor der endgültigen Integration eingearbeitet werden müssen.

Viele tag-tägliche Aufgaben lassen sich mit Hilfe der Boost-Bibliotheken einfach, elegant und portabel lösen. Darum sollten Sie, wenn Sie ein Problem haben und in der C++ Standard-Bibliothek nicht fündig werden, zuerst bei Boost nach einer fertigen Lösung schauen.

4.6 Dokumentation

Wenn Sie mit C++ entwickeln, so werden Sie auch eine Dokumentation der Sprache und der Standard-Bibliothek benötigen. Für den Einstieg in C++, für die Beispiele und Aufgaben enthält das Buch selber genügend Informationen. Aber natürlich gibt es Themen und Fragen, die hier nicht beantwortet werden. Zum Glück gibt es heute das Internet, in dem Sie viele Informationen finden.

Für allgemeine Informationen zu C++ und die Standardisierung empfiehlt sich die offizielle Seite der ISO C++ Gruppe. Hier finden Sie neben einem ersten Überblick auch viele Details zum Stand der Standardisierung, Links zu aktuellen Blogs und Videos, und andere Neuigkeiten rund um C++.

- http://isocpp.org/

Wenn Sie eine Referenz der Sprache C++ und der Standard-Bibliothek benötigen – und ja, Sie werden sie brauchen –, dann gibt es zwei sehr empfehlenswerte Seiten im Internet:

- http://en.cppreference.com/w/
- http://www.cplusplus.com/
- http://www.cplusplus.com/reference/

Wenn Sie Fragen und Probleme mit C++ haben, oder Ihre Programme partout nicht laufen oder funktionieren wollen, und Sie keinen Freund oder Kollegen fragen können – dann verzweifeln Sie nicht zu lange, sondern fragen Sie im Internet nach. Die mit Abstand wichtigste Frage/Antwort Plattform für Programmierer im Internet ist „StackOverflow". Es gibt kaum eine Programmier-Frage, die in StackOverflow nicht schon gestellt und beantwortet wurde. Und wenn doch: Die Erfahrung zeigt, dass meist schon nach wenigen Minuten mehrere sehr hilfreiche Antworten vorhanden sind. Auch Google liefert bei Programmier-Fragen fast immer StackOverflow Antworten als die ersten Treffer. Also verzweifeln Sie nicht, sondern fragen Sie dort nach.

- https://stackoverflow.com/
- https://stackoverflow.com/questions/tagged/c%2b%2b

4.7 Installation Microsoft Visual Studio

Dieses Kapitel beschreibt kurz die Installation des Microsoft Visual Studios 2022 in der Community Edition. Alle Beschreibungen und Screenshots in diesem Buch beziehen sich auf die Version 17.2.5. Wenn Sie das Buch in den Händen halten, gibt es schon eine neuere Version des Microsoft Visual Studios, da Microsoft ständig Updates herausgibt. Vom Grundsatz her werden die Beschreibungen und Screenshots aber noch stimmen, solange nicht eine komplett neue Version des Microsoft Visual Studios herauskommt.

Laden Sie den Installer für das Microsoft Visual Studio Community 2022 von der Microsoft Web-Seite herunter: „https://visualstudio.microsoft.com/de/vs/community/“. Achten Sie darauf, dass Sie die Community Edition herunterladen – alle anderen Varianten des Microsoft Visual Studios sind kostenpflichtig.

Hinweis: Alle Screenshots sind Mitte 2022 entstanden. Wenn Sie dieses Buch in den Händen halten, gibt es schon neuere Versionen des Visual Studios. Im Detail könnte der Installations-Vorgang – insbesondere der Auswahl-Dialog – anders aussehen. Das sollte Sie aber nicht stören. Installieren Sie sich immer die neuste Version.

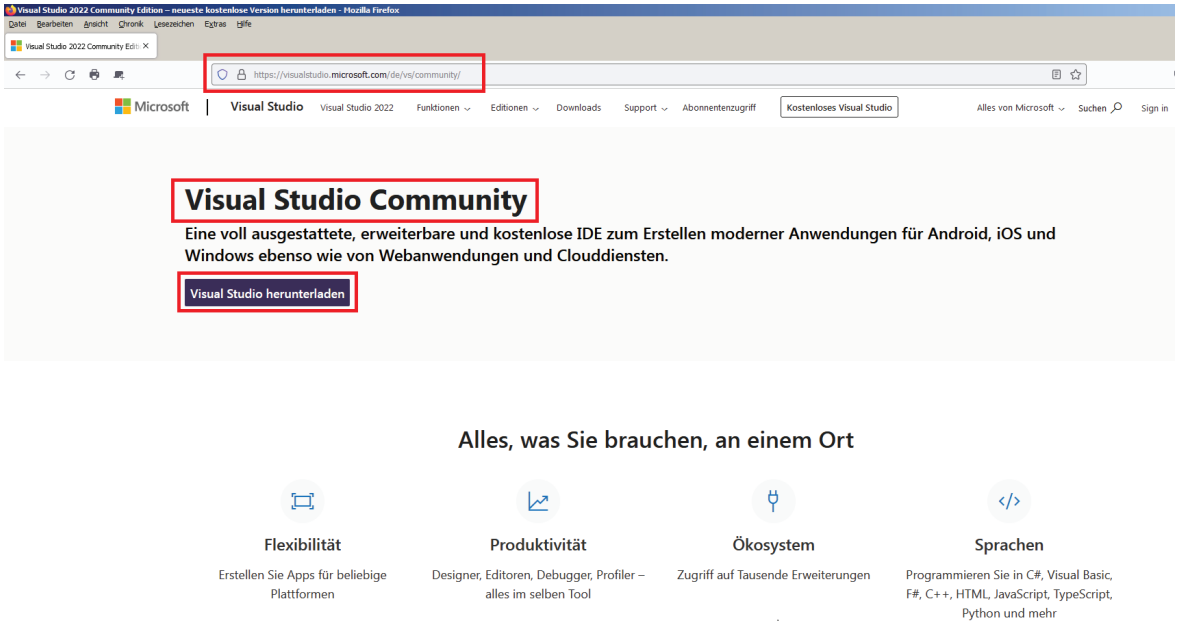


Abb. 4-7 Microsoft Web-Seite für das Microsoft Visual Studio Community

Hinweis: Sie laden hier nicht das Microsoft Visual Studio selber herunter, sondern nur einen initialen Installer, der dann erst selber das Microsoft Visual Studio herunterlädt und installiert. Darum ist dieser Installer auch nur etwas über 1 MByte groß. Für eine IDE mit Compiler ist das viel zu wenig.

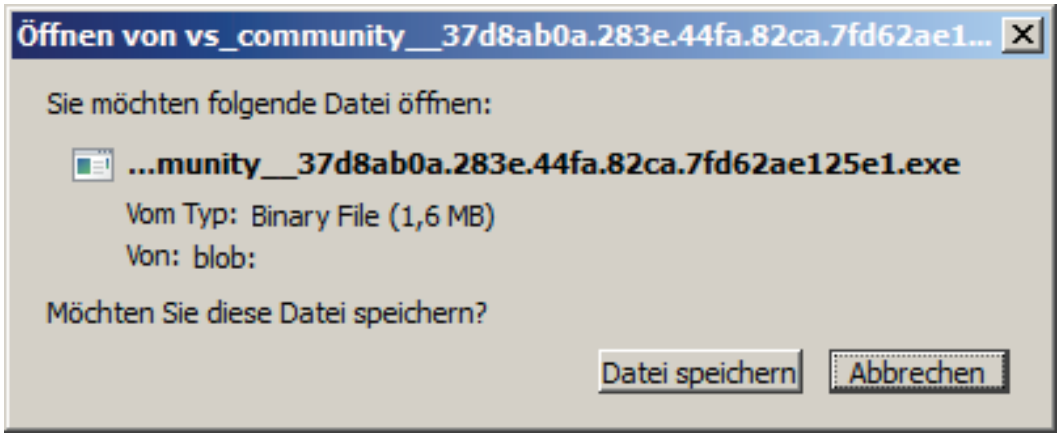


Abb. 4-8 : Download des Microsoft Visual Studio Community Installers

Speichern Sie den Installer und führen Sie ihn aus.

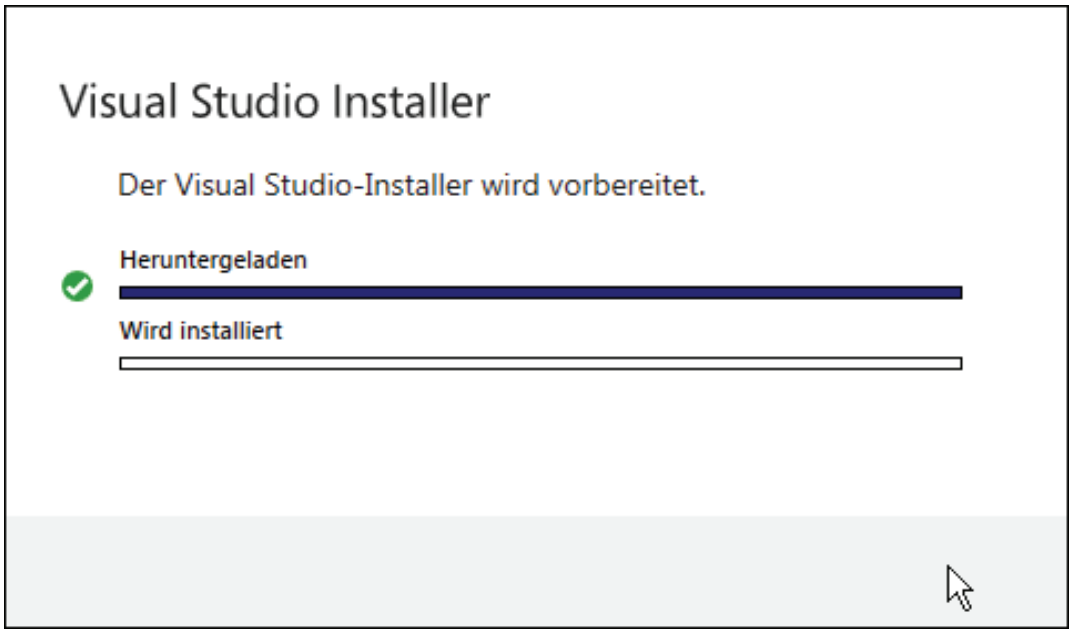


Abb. 4-9 Microsoft Visual Studio Community Installer in Arbeit

Nach einer kurzen Weile bekommen Sie folgenden Auswahl-Dialog. Hier können Sie definieren, für welche Ziel-Plattformen und mit welcher Programmier-Sprache Sie im Microsoft Visual Studio entwickeln möchten. Sie können später jederzeit einzelne Module nachinstallieren oder wieder löschen. Achtung: Wichtig ist hier, dass Sie die „Desktop-entwicklung mit C++“ auswählen – denn das wollen wir ja machen.

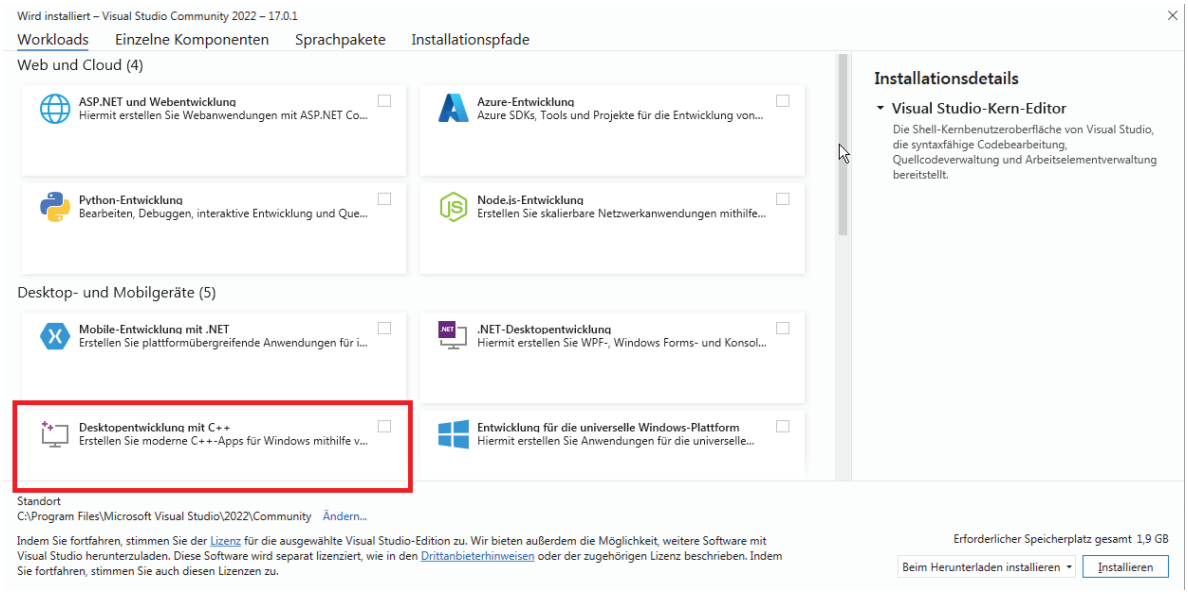


Abb. 4-10 Microsoft Visual Studio Installations Auswahl-Dialog

Mit dem Button „Installieren“ starten Sie endlich den eigentlichen Installations-Vorgang. Jetzt wird das eigentliche Microsoft Visual Studio Community 2022 heruntergeladen und installiert. Aufgrund der Größe kann dies eine gewisse Zeit dauern.

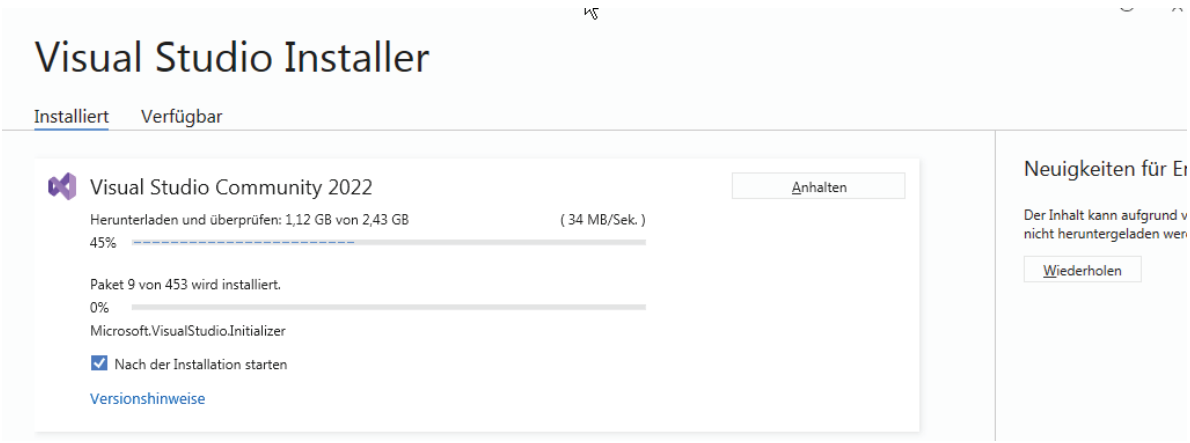


Abb. 4-11 Microsoft Visual Studio Community Installer in Arbeit

Am Ende des Installations-Vorgangs werden Sie aufgefordert, den Rechner neu zu starten. Machen Sie dies. Vorher sollten Sie das Microsoft Visual Studio nicht benutzen. Teile des Microsoft Visual Studios, die wir in diesem Buch nicht besprechen werden, wie zum Beispiel der Debugger, müssen sich recht tief im System verankern, um korrekt zu arbeiten. Daher muss der Neustart erfolgen.

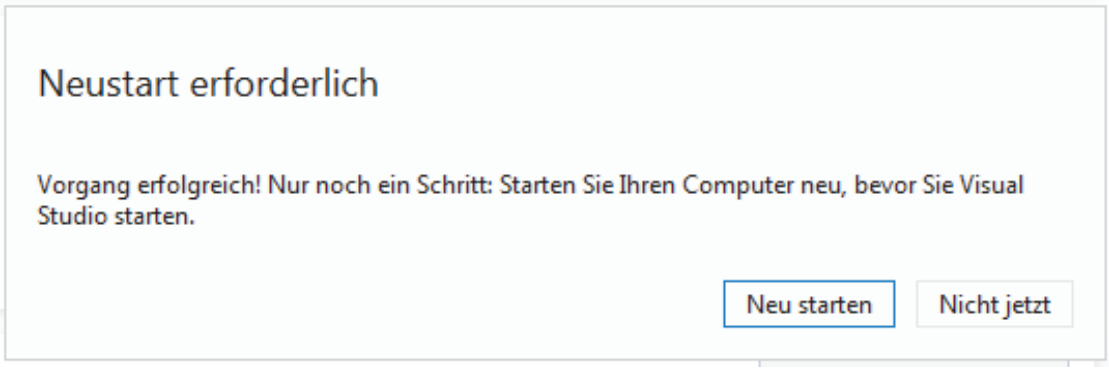


Abb. 4-12 Neustart des Rechners am Ende der Installation

Mit dem Neustart des Rechners steht uns das Microsoft Visual Studio als Integrierte Entwicklungsumgebung (IDE) zur Verfügung, und wir können nach der Anmeldung bei den Microsoft Entwicklerdiensten endlich mit dem eigentlichen C++ Programmieren beginnen.

4.8 Microsoft Entwicklerdienste

Wenn Sie das Microsoft Visual Studio zum ersten Mal starten, dann erscheint nach dem Startbildschirm folgender Anmelde-Dialog für die Microsoft Entwicklerdienste.

Abb. 4-13 Anmelde-Dialog für die Microsoft Entwicklerdienste

Ohne eine Anmeldung bei den Microsoft Entwicklerdiensten läuft das Microsoft Visual Studio nur mit einer Test-Lizenz, die zwar keine funktionellen Einschränkungen hat, aber auf 30 Tage begrenzt ist. Spätestens nach 30 Tagen müssen Sie sich also anmelden, oder mit einer anderen IDE oder einem anderen Editor arbeiten. Für die Anmeldung benötigen Sie nur einen E-Mail Account bei einem der Microsoft Dienste, wie zum Beispiel „Bing“ oder „Outlook“. Sie müssen auch nichts weiter mit dem Konto machen oder verbinden. Ich zum Beispiel habe mir bei „outlook.de“ einen E-Mail Account extra für das Microsoft Visual Studio angelegt und benutze es für nichts anderes als hierfür. Also erstellen Sie ein entsprechendes Konto – geht direkt im Dialog – und dann können Sie das Microsoft Visual Studio nutzen, und wir können endlich mit C++ starten.