

Inhalt

1	Grundlagen	1
1.1	Unser erstes Programm	1
1.2	Variablen	3
1.2.1	Fundamentale Typen	4
1.2.2	Characters und Strings	5
1.2.3	Variablen deklarieren	6
1.2.4	Konstanten	6
1.2.5	Literale	7
1.2.6	Werterhaltende Initialisierung	9
1.2.7	Gültigkeitsbereiche	10
1.3	Operatoren	12
1.3.1	Arithmetische Operatoren	13
1.3.2	Boolesche Operatoren	15
1.3.3	Bitweise Operatoren	16
1.3.4	Zuweisung	17
1.3.5	Programmablauf	18
1.3.6	Speicherverwaltung	19
1.3.7	Zugriffsoperatoren	19
1.3.8	Typbehandlung	19
1.3.9	Fehlerbehandlung	20
1.3.10	Überladung	20
1.3.11	Operatorprioritäten	20
1.3.12	Vermeiden Sie Seiteneffekte!	21
1.4	Ausdrücke und Anweisungen	23
1.4.1	Ausdrücke	23
1.4.2	Anweisungen	24
1.4.3	Verzweigung	24
1.4.4	Schleifen	27
1.4.5	goto	30
1.5	Funktionen	31
1.5.1	Argumente	31

1.5.2	Rückgabe der Ergebnisse.....	33
1.5.3	Inlining	34
1.5.4	Überladen	34
1.5.5	Die main-Funktion	36
1.6	Fehlerbehandlung	37
1.6.1	Zusicherungen.....	37
1.6.2	Ausnahmen	39
1.6.3	Statische Zusicherungen	43
1.7	I/O	44
1.7.1	Standard-Ausgabe.....	44
1.7.2	Standard-Eingabe	45
1.7.3	Ein-/Ausgabe mit Dateien	45
1.7.4	Generisches Stream-Konzept.....	46
1.7.5	Formatierung	47
1.7.6	I/O-Fehler behandeln	48
1.7.7	File-System.....	51
1.8	Arrays, Zeiger und Referenzen	52
1.8.1	Arrays	52
1.8.2	Zeiger	54
1.8.3	Intelligente Zeiger	57
1.8.4	Referenzen	60
1.8.5	Vergleich zwischen Zeigern und Referenzen	60
1.8.6	Nicht auf abgelaufene Daten verweisen!.....	61
1.8.7	Containers for Arrays	62
1.9	Strukturierung von Software-Projekten	64
1.9.1	Kommentare	64
1.9.2	Präprozessor-Direktiven	66
1.10	Aufgaben.....	70
1.10.1	Verengung	70
1.10.2	Literale	70
1.10.3	Operatoren	70
1.10.4	Verzweigung	70
1.10.5	Schleifen.....	70
1.10.6	I/O.....	71
1.10.7	Arrays und Zeiger.....	71
1.10.8	Funktionen.....	71

2	Klassen	72
2.1	Universell programmieren, nicht detailversessen	72
2.2	Member	74
2.2.1	Mitgliedervariablen	75
2.2.2	Zugriffsrechte	75
2.2.3	Zugriffsoperatoren	78
2.2.4	Der static-Deklarator für Klassen	78
2.2.5	Member-Funktionen	79
2.3	Konstruktoren und Zuweisungen	80
2.3.1	Konstruktoren	80
2.3.2	Zuweisungen	90
2.3.3	Initialisierungslisten	91
2.3.4	Einheitliche Initialisierung	93
2.3.5	Move-Semantik	95
2.3.6	Objekte aus Literalen konstruieren	101
2.4	Destruktoren	103
2.4.1	Implementierungsregeln	104
2.4.2	Richtiger Umgang mit Ressourcen	104
2.5	Zusammenfassung der Methodengenerierung	110
2.6	Zugriff auf Mitgliedervariablen	111
2.6.1	Zugriffsfunktionen	111
2.6.2	Index-Operator	112
2.6.3	Konstante Mitgliederfunktionen	113
2.6.4	Referenz-qualifizierte Mitglieder	115
2.7	Design von Operatorüberladung	116
2.7.1	Seien Sie konsistent!	116
2.7.2	Die Priorität respektieren	117
2.7.3	Methoden oder freie Funktionen	118
2.8	Aufgaben	121
2.8.1	Polynomial	121
2.8.2	Rational	121
2.8.3	Move-Zuweisung	122
2.8.4	Initialisierungsliste	122
2.8.5	Ressourcenrettung	122

3	Generische Programmierung	123
3.1	Funktions-Templates	123
3.1.1	Parametertyp-Deduktion	124
3.1.2	Mit Fehlern in Templates klarkommen	128
3.1.3	Gemischte Typen	129
3.1.4	Einheitliche Initialisierung	130
3.1.5	Automatischer Rückgabetyt	130
3.2	Namensräume und Funktionssuche	131
3.2.1	Namensräume	131
3.2.2	Argumentabhängiges Nachschlagen	134
3.2.3	Namensraum-Qualifizierung oder ADL	138
3.3	Klassen-Templates	140
3.3.1	Ein Container-Beispiel	140
3.3.2	Einheitliche Klassen- und Funktionsschnittstellen entwerfen	142
3.4	Typ-Deduktion und -Definition	148
3.4.1	Automatische Variablentypen	148
3.4.2	Typ eines Ausdrucks	148
3.4.3	<code>decltype(auto)</code>	149
3.4.4	Deduzierte Klassen-Template-Parameter	151
3.4.5	Mehrere Typen deduzieren	152
3.4.6	Typen definieren	153
3.5	Etwas Theorie zu Templates: Konzepte	155
3.6	Template-Spezialisierung	156
3.6.1	Spezialisierung einer Klasse für einen Typ	156
3.6.2	Funktionen spezialisieren und Überladen	158
3.6.3	Partielle Spezialisierung von Klassen	160
3.6.4	Partiell spezialisierte Funktionen	161
3.6.5	Strukturierte Bindung mit Nutzertypen	163
3.7	Nicht-Typ-Parameter für Templates	166
3.7.1	Container fester Größe	166
3.7.2	Nicht-Typ-Parameter deduzieren	169
3.8	Funktoren	169
3.8.1	Funktionsartige Parameter	171
3.8.2	Funktoren zusammensetzen	172
3.8.3	Rekursion	174
3.8.4	Generische Reduktion	177
3.9	Lambdas	178
3.9.1	Objekte erfassen	179

3.9.2	Generische Lambdas	183
3.10	Variablen-Templates	183
3.11	Variadische Templates	185
3.11.1	Rekursive Funktionen	185
3.11.2	Direkte Expansion	187
3.11.3	Indexsequenzen	188
3.11.4	Faltung	190
3.11.5	Typgeneratoren	191
3.11.6	Wachsende Tests	191
3.12	Übungen	193
3.12.1	String-Darstellung	193
3.12.2	String-Darstellung von Tupeln	193
3.12.3	Generischer Stack	194
3.12.4	Rationale Zahlen mit Typparameter	194
3.12.5	Iterator eines Vektors	194
3.12.6	Ungerader Iterator	194
3.12.7	Bereich von ungeraden Zahlen	195
3.12.8	Stack von bool	195
3.12.9	Stack mit nutzerdefinierter Größe	195
3.12.10	Deduktion von Nicht-Typ-Template-Argumenten	195
3.12.11	Trapez-Regel	196
3.12.12	Partielle Spezialisierung mit einer statischen Funktion	196
3.12.13	Funktor	196
3.12.14	Lambda	196
3.12.15	Implementieren Sie <code>make_unique</code>	197

4 Bibliotheken **198**

4.1	Standard-Template-Library	199
4.1.1	Einführendes Beispiel	199
4.1.2	Iteratoren	200
4.1.3	Container	205
4.1.4	Algorithmen	214
4.1.5	Jenseits von Iteratoren	219
4.1.6	Parallele Berechnung	221
4.2	Numerik	222
4.2.1	Komplexe Zahlen	222
4.2.2	Zufallszahlengeneratoren	225
4.2.3	Mathematische Spezialfunktionen	234

- 4.3 Meta-Programmierung 235
 - 4.3.1 Wertgrenzen 235
 - 4.3.2 Typeigenschaften..... 237
- 4.4 Utilities 239
 - 4.4.1 optional 239
 - 4.4.2 Tupel 240
 - 4.4.3 variant 243
 - 4.4.4 any 245
 - 4.4.5 string_view..... 246
 - 4.4.6 function 247
 - 4.4.7 Referenz-Wrapper 250
- 4.5 Die Zeit ist gekommen 252
- 4.6 Parallelität 254
 - 4.6.1 Terminologie 254
 - 4.6.2 Überblick 255
 - 4.6.3 Threads 255
 - 4.6.4 Rückmeldung an den Aufrufer 257
 - 4.6.5 Asynchrone Aufrufe 258
 - 4.6.6 Asynchroner Gleichungslöser 260
 - 4.6.7 Variadische Mutex-Sperre 264
- 4.7 Wissenschaftliche Bibliotheken jenseits des Standards 266
 - 4.7.1 Andere Arithmetiken 266
 - 4.7.2 Intervallarithmetik 267
 - 4.7.3 Lineare Algebra 267
 - 4.7.4 Gewöhnliche Differentialgleichungen 268
 - 4.7.5 Partielle Differentialgleichungen..... 268
 - 4.7.6 Graphenalgorithmien 269
- 4.8 Übungen 269
 - 4.8.1 Sortierung nach Betrag 269
 - 4.8.2 Suche mit einem Lambda als Prädikat 269
 - 4.8.3 STL-Container 270
 - 4.8.4 Komplexe Zahlen..... 270
 - 4.8.5 Parallele Vektoraddition 271
 - 4.8.6 Refaktorisierung der parallelen Addition 271

5	Meta-Programmierung	273
5.1	Lassen Sie den Compiler rechnen	273
5.1.1	Kompilierzeitfunktionen	273
5.1.2	Erweiterte Kompilierzeitfunktionen	275
5.1.3	Primzahlen	277
5.1.4	Wie konstant sind unsere Konstanten?	279
5.1.5	Kompilierzeit-Lambdas	280
5.2	Typinformationen	281
5.2.1	Typabhängige Funktionsergebnisse	281
5.2.2	Bedingte Ausnahmebehandlung	285
5.2.3	Ein Beispiel für eine const-korrekte View	286
5.2.4	Standard-Typmerkmale	293
5.2.5	Domän-spezifische Type-Traits	293
5.2.6	Typeigenschaften mit Überladung	295
5.2.7	enable_if	297
5.2.8	Variadische Templates überarbeitet	301
5.3	Expression-Templates	304
5.3.1	Einfache Implementierung eines Additionsoperators	304
5.3.2	Eine Klasse für Expression-Templates	308
5.3.3	Generische Expression-Templates	310
5.4	Compiler-Optimierung mit Meta-Tuning	312
5.4.1	Klassisches Abrollen mit fester Größe	313
5.4.2	Geschachteltes Abrollen	317
5.4.3	Aufwärmung zum dynamischen Abrollen	323
5.4.4	Abrollen von Vektorausdrücken	324
5.4.5	Tuning von Expression-Templates	326
5.4.6	Tuning von Reduktionen	329
5.4.7	Tuning geschachtelter Schleifen	336
5.4.8	Resümee des Tunings	341
5.5	Turing-Vollständigkeit	343
5.6	Übungen	346
5.6.1	Type-Traits	346
5.6.2	Fibonacci-Sequenz	346
5.6.3	Meta-Programm für den größten gemeinsamen Divisor	346
5.6.4	Rationale Zahlen mit gemischten Typen	347
5.6.5	Vektor-Expression-Template	347
5.6.6	Meta-Liste	348

6	Objektorientierte Programmierung	349
6.1	Grundprinzipien	349
6.1.1	Basis- und abgeleitete Klassen	350
6.1.2	Konstruktoren erben	353
6.1.3	Virtuelle Funktionen	354
6.1.4	Funktoren über Vererbung	361
6.1.5	Abgeleitete Klassen für Ausnahmen	362
6.2	Redundanz entfernen	363
6.3	Mehrfachvererbung	365
6.3.1	Mehrere Eltern	365
6.3.2	Gemeinsame Großeltern	366
6.4	Dynamische Auswahl von Subtypen	371
6.5	Konvertierung	373
6.5.1	Umwandlungen zwischen abgeleiteten Klassen	374
6.5.2	<code>const_cast</code>	378
6.5.3	Umdeutung	379
6.5.4	Umwandlung im Funktionsstil	379
6.5.5	Implizite Umwandlungen	381
6.6	CRTP	382
6.6.1	Ein einfaches Beispiel	382
6.6.2	Ein wiederverwendbarer Indexoperator	384
6.7	Übungen	386
6.7.1	Nicht-redundante Raute	386
6.7.2	Vektorklasse mit Vererbung	386
6.7.3	Ausnahmen in Vektor refaktorisieren	386
6.7.4	Test auf geworfene Ausnahme	387
6.7.5	Klonfunktion	387
7	Wissenschaftliche Projekte	388
7.1	Implementierung von ODE-Lösern	388
7.1.1	Gewöhnliche Differentialgleichungen	388
7.1.2	Runge-Kutta-Algorithmen	391
7.1.3	Generische Implementierung	392
7.1.4	Ausblick	399
7.2	Projekte erstellen	399
7.2.1	Build-Prozess	400
7.2.2	Build-Tools	404
7.2.3	Separates Kompilieren	408
7.3	Einige abschließende Worte	414

A	Weitschweifendes	416
A.1	Mehr über gute und schlechte Software	416
A.2	Grundlagen im Detail	422
A.2.1	Statische Variablen	422
A.2.2	Mehr über <code>if</code>	422
A.2.3	Duff's Device	424
A.2.4	Programmaufrufe	424
A.2.5	Zusicherung oder Ausnahme?	425
A.2.6	Binäre I/O	426
A.2.7	I/O im Stile von C	427
A.2.8	Garbage-Collection	428
A.2.9	Ärger mit Makros	429
A.3	Praxisbeispiel: Matrix-Invertierung	430
A.4	Klassendetails	440
A.4.1	Zeiger auf Mitglieder	440
A.4.2	Weitere Initialisierungsbeispiele	440
A.4.3	Zugriff auf mehrdimensionale Datenstrukturen	441
A.5	Methodengenerierung	444
A.5.1	Automatische Generierung	445
A.5.2	Steuerung der Generierung	447
A.5.3	Generierungsregeln	448
A.5.4	Fallstricke und Designrichtlinien	452
A.6	Template-Details	456
A.6.1	Einheitliche Initialisierung	456
A.6.2	Welche Funktion wird aufgerufen?	456
A.6.3	Spezialisierung auf spezifische Hardware	459
A.6.4	Variadisches binäres I/O	460
A.7	Mehr über Bibliotheken	461
A.7.1	<code>std::vector</code> in C++03 verwenden	461
A.7.2	<code>variant</code> mal nerdisch	462
A.8	Dynamische Auswahl im alten Stil	462
A.9	Mehr über Meta-Programmierung	463
A.9.1	Das erste Meta-Programm in der Geschichte	463
A.9.2	Meta-Funktionen	465
A.9.3	Rückwärtskompatible statische Zusicherung	466
A.9.4	Anonyme Typparameter	467
A.9.5	Benchmark-Quellen für dynamisches Abrollen	471
A.9.6	Benchmark für Matrixprodukt	472

B	Werkzeuge	473
	B.1 g++	473
	B.2 Debugging	474
	B.2.1 Textbasierte Debugger	474
	B.2.2 Debugging mit graphischen Interface: DDD	476
	B.3 Speicheranalyse	478
	B.4 gnuplot	479
	B.5 Unix, Linux und Mac OS	480
C	Sprachdefinitionen	482
	C.1 Wertkategorien	482
	C.2 Konvertierungsregeln	485
	C.2.1 Aufwertung	486
	C.2.2 Andere Konvertierungen	486
	C.2.3 Arithmetische Konvertierungen	487
	C.2.4 Verengung	488
	Literatur	489
	Abbildungsverzeichnis	492
	Tabellenverzeichnis	493
	Index	494