

Jürgen Lemke

Xpert.press

C++ - Metaprogrammierung

Eine Einführung in die
Präprozessor- und Template-
Metaprogrammierung

 Springer Vieweg

Xpert.press

Weitere Information zu dieser Reihe finden Sie auf
<http://www.springer.com/series/4393>

Die Reihe **Xpert.press** vermittelt Professionals in den Bereichen Softwareentwicklung, Internettechnologie und IT-Management aktuell und kompetent relevantes Fachwissen über Technologien und Produkte zur Entwicklung und Anwendung moderner Informationstechnologien.

Jürgen Lemke

C++-Metaprogrammierung

Eine Einführung in die Präprozessor- und
Template-Metaprogrammierung

Jürgen Lemke
Hilchenbach, Deutschland

ISSN 1439-5428

Xpert.press

ISBN 978-3-662-48549-1

ISBN 978-3-662-48550-7 (eBook)

DOI 10.1007/978-3-662-48550-7

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

Springer Vieweg

© Springer-Verlag Berlin Heidelberg 2016

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

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 und daher von jedermann benutzt werden dürften.

Der Verlag, die Autoren und die Herausgeber gehen davon aus, dass die Angaben und Informationen in diesem Werk zum Zeitpunkt der Veröffentlichung vollständig und korrekt sind. Weder der Verlag, noch die Autoren oder die Herausgeber übernehmen, ausdrücklich oder implizit, Gewähr für den Inhalt des Werkes, etwaige Fehler oder Äußerungen.

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer-Verlag GmbH Berlin Heidelberg ist Teil der Fachverlagsgruppe Springer Science+Business Media (www.springer.com)

*Dieses Buch widme ich meiner Frau Heike sowie
unseren Kindern Annelie, Frieda und Torsten.*

Vorwort

Kann man eine Programmiersprache wie C++ mit seinen Templates komplett beherrschen? Selbst Bjarne Stroustrup, der Erfinder von C++, musste sich vom Template-Mechanismus der Turing-Vollständigkeit eines Besseren belehren lassen. Auf dem C++-Standardisierungstreffen 1994 in SanDiego präsentierte Erwin Unruh zum ersten Mal ein rekursives C++-Metaprogramm zur Berechnung von Primzahlen.

Gibt es heute noch Grenzen von C++, wo liegen sie und was ist mit C++ möglich bzw. unmöglich? In meiner Arbeit mit C++-Schnittstellen (Interfaces) wurde ich mit der Behauptung konfrontiert, dass sich atomare (native) C++-Strukturen nicht automatisch innerhalb der Programmiersprache generieren lassen, im Gegensatz zu managed Strukturen von Java, C# und C++/CLI. Weil ich diese Aussage so nicht im Raum stehen lassen wollte, beschäftigte ich mich intensiv mit der Metaprogrammierung in C++. Das Ergebnis ist ein Konzept zur automatischen Generierung von atomaren Strukturen, einschließlich deren Serialisierung und Visualisierung, sowie eine große Sammlung von Template-Metafunktionen.

Doch wie kann nun der neue Stoff dem Leser vermittelt werden? Ein Nachschlagewerk für die C++-Metaprogrammierung zu schreiben ist kaum möglich, da die Metaprogrammierung stärker mit Programmier Techniken als mit C++-Sprachstandards verbunden ist. Eine Beschränkung auf die Darstellung von Programmier Techniken schränkt wiederum den Leserkreis zu stark ein, da bestimmte Vorkenntnisse der Präprozessor- und Templateprogrammierung vorhanden sein müssen. Aus diesem Grunde versuche ich einen Spagat zwischen beiden Ansätzen zu erreichen.

In den ersten Abschnitten des Buches werden die Grundlagen in Form eines Nachschlagewerkes sowohl für den Präprozessor als auch für den Templatemechanismus beschrieben. Diese Beschreibung zielt bereits auf die spätere Verwendung in der Metaprogrammierung ab, sodass diese Abschnitte in der Beschreibung der Grundlagen

bereits deutlich umfangreicher sind, als sie in Standard C++-Büchern sein können. In den weiteren Abschnitten werden dann Programmier Techniken und Anwendungsbeispiele vorgestellt, die auf die Grundlagenabschnitte aufbauen. Viele dieser Beispiele und Techniken ziehen sich dann wie ein roter Faden durch das gesamte Buch.

Bei der Untersuchung von Laufzeitfehlern fällt häufig auf, dass eigentlich schon der Compiler auf bestimmte Fehler hätte aufmerksam machen müssen, weil alle Informationen bereits zur Kompilationszeit zur Verfügung standen. Leider gehen zum Beispiel Längeninformationen von Feldern oder Typinformationen von Objekten bei der Parameterübergabe in Funktionen verloren. Mit Templates lassen sich solche Informationen in andere Programmcodes übertragen. Ein Schwerpunkt der Metaprogrammierung liegt daher auch in der Erzeugung von sicherem Programmcode.

Der interessierte Leser wird nach dem Studium des Buches einen erweiterten Blickwinkel auf die Möglichkeiten des C++-Compilers, der Sicherheit und der Effizienz von Programmcode erhalten. Er wird schon vor dem eigentlichen Schreiben von neuem Programmcode tiefgreifende Überlegungen anstellen, welche Programmteile generiert und welche Teile bereits zur Kompilationszeit erledigt werden können.

Auch dem .NET-Anwender steht die C++-Metaprogrammierung mit seinen Templates, über die erweiterte Programmiersprache C++/CLI, zur Verfügung. Damit können .NET-Anwendungen im Hinblick auf ihre Laufzeit beschleunigt werden. Mit Hilfe von Generatoren ist es leicht möglich, Strukturen nach .NET zu konvertieren und die Daten von unmanaged zu managed Objekten zu kopieren bzw. zu konvertieren.

In der vorliegenden Arbeit dreht sich aber nicht alles nur um die reine Metaprogrammierung. Das ist auch kaum möglich, denn die Metaprogrammierung ist immer auf Hilfsklassen angewiesen. Gerade dieses Zusammenspiel soll dargestellt werden, um den Leser für eigene Ansätze zu inspirieren.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Ziel und Zweck des Buches	2
1.2	Vergleich zur Codegenerierung außerhalb von C++	2
1.3	Syntax der verwendeten Beispiele	3
	Literatur	8
 Teil I Grundlagen		
2	C-Präprozessor	11
2.1	Einbinden von Dateiinhalten	12
2.2	Einfache Makros	12
2.3	Makros mit Parametern	13
2.4	Mehrzeilige Makros	15
2.5	Umwandlung und Bearbeitung von Zeichenketten	16
2.6	Bedingte Ersetzung	17
2.7	Iterationen mit Makros	18
	2.7.1 Einfache horizontale Iterationen	18
	2.7.2 Iterationen mit #include	20
2.8	Variadische Makros	21
	2.8.1 Einführung in variadische Makros	21
	2.8.2 Bestimmung der Anzahl der variablen Argumente	23
	2.8.3 Iterationen mit variadischen Argumenten	27
	2.8.4 Leere Parameter identifizieren	29
2.9	Rechnen mit Makroparametern	30
	2.9.1 Basismakros zum Inkrementieren und Dekrementieren	30
	2.9.2 Addition und Subtraktion	32
	2.9.3 Multiplikation und Division	34
	2.9.4 Bestimmung von Primzahlen mit Makros	36
2.10	Compiler-Direktiven	39
	Literatur	40

3	Templates	41
3.1	Funktionstemplates	42
3.1.1	Deklaration	42
3.1.2	Nichttyp-Parameter	44
3.1.3	Reihenfolge der Templateargumente	45
3.1.4	Überladung von Funktionstemplates	45
3.1.5	Vollständige Spezialisierung von Funktionstemplates	47
3.1.6	Rekursiver Aufruf von Funktionstemplates	48
3.1.7	Indirekte partielle Spezialisierung	54
3.2	Klassentemplates	58
3.2.1	Deklaration	58
3.2.2	Nichttyp-Parameter	59
3.2.3	Standardwerte für Templateargumente	60
3.2.4	Vollständige Spezialisierung von Klassentemplates	61
3.2.5	Partielle Spezialisierung von Klassentemplates	61
3.2.6	Rekursiver Aufruf von Klassentemplates	63
3.2.7	Die Verwendung von Enumeratoren	64
3.2.8	Statische If-Bedingung und Switch-Anweisung	65
3.2.9	Der this-Zeiger	69
3.3	Template Template Parameter	70
3.4	Strings als Argumente für Templateparameter	74
3.5	Variadische Templates in C++11	76
3.5.1	Deklaration	76
3.5.2	Variadische Funktionstemplates	77
3.5.3	Variadische Klassentemplates	78
3.5.4	Metaprogrammierung mit variadischen Templates	81
3.6	Das Schlüsselwort <code>constexpr</code> ab C++11	85
	Literatur	87
4	Erweiterte Metaprogrammierung	89
4.1	Typlisten	89
4.1.1	Definition von Typlisten	89
4.1.2	Arbeiten mit Typlisten	93
4.2	Type-Traits	97
4.3	SFINAE	105
4.3.1	Das SFINAE-Prinzip	105
4.3.2	Der <code>sizeof</code> -Trick	107
4.3.3	Gruppierung von überladenen Memberfunktionen	113
4.3.4	Testen auf Software-Updates	115
4.4	Weitere Traits mit <code>sizeof</code>	118
4.5	Assertion zur Kompilationszeit	119
4.6	Umgehen von tiefen Rekursionen	123
4.7	Neue Metafunktionen der STL in C++11 (Type-Traits)	126
	Literatur	131

Teil II Techniken und Anwendungsfälle

5	Sichere Schnittstellen (Interfaces)	135
5.1	Parameterprüfung zur Kompilationszeit	135
5.1.1	Prüfen von Nichttyp-Templateparametern	135
5.1.2	Prüfen von Typ-Parametern	141
5.2	Konfigurationstemplates	144
5.2.1	Konfiguration variabler Typen	145
5.2.2	Konfiguration von veränderlichen Schnittstellen	146
5.2.3	Generierung von Fabrikklassen	148
	Literatur	155
6	C++-Metaprogrammierung für Strukturen	157
6.1	Einführendes Beispiel zur Generierung von Strukturen	157
6.2	Generieren von Strukturen mit Makros	158
6.2.1	Generierung von einfachen Strukturen	158
6.2.2	Erweiterte statische Funktionen	161
6.2.3	Vereinfachungen mit variadischen Makros	164
6.3	Einsatz von Typlisten	164
6.3.1	Anlegen der Typliste	164
6.3.2	Generierung von Strukturen mittels Typlisten	166
6.3.3	Konvertieren von Typlisten	167
6.4	Verwaltungsklassen für Strukturen	170
6.4.1	Klassengeneratoren für Typlisten	170
6.4.2	Basisklasse für alle Typlisten	172
6.4.3	Angepasste Typen für Klassengeneratoren	174
6.4.4	Weitere Optimierung der Strukturdefinition	178
6.4.5	Serialisierung der Struktur	180
6.4.6	Spezialisierung der Verwaltungsklassen	188
6.5	Zugriff auf Elemente der Struktur	189
6.5.1	Zugriff über Index	189
6.5.2	Zugriff über Index zur Kompilationszeit	194
6.5.3	Zugriff über Typnamen	195
6.5.4	Zuweisung mit Kommaoperator	197
6.6	Konvertieren von C++ nach C++/CLI und umgekehrt	200
6.6.1	Konvertieren der Typliste	202
6.6.2	Generieren der C++/CLI-Struktur	206
6.6.3	Kopieren der Daten von C++ nach C++/CLI und umgekehrt	207
6.7	Dokumentation der generierten Strukturen	212
6.7.1	Vorbetrachtungen	212
6.7.2	Erweiterung der Verwaltungsklasse um die Dokumentation	214

6.7.3	Dokumentation mit Typlisten	217
6.7.4	Schreiben der Dokumentation für beliebige Generatoren	225
	Literatur	232
7	Weitere Anwendungsbeispiele für generierte Strukturen	233
7.1	Allgemeine Vorgehensweise	233
7.2	Generierung einer Datenbankschnittstelle	234
7.2.1	Verwaltungsklasse zur Datenbankanbindung	234
7.2.2	Anlegen einer Datenbanktabelle	238
7.2.3	Lesen und Schreiben von Daten	243
7.2.4	Ausblick auf erweiterte Funktionalitäten	248
7.3	Generierung einer Visualisierung	250
7.3.1	Generierung von graphischen Dialogen mit wxWidgets	250
7.3.2	Generierung von graphischen Dialogen mit Qt	255
	Literatur	259
8	Sicheres Rechnen mit Einheiten	261
8.1	Vorbetrachtungen	261
8.2	Einheiten mit Templates realisieren	266
8.3	Das SI-Einheitensystem	269
8.4	Vereinfachte Einheitensysteme	277
8.4.1	Das MKS-Einheitensystem	277
8.4.2	Ein Ingenieur-Einheitensystem	279
8.4.3	Das astronomische Einheitensystem	281
8.5	Konvertierung zwischen den Einheitensystemen	284
8.6	Ein flexibles Einheitensystem	285
8.6.1	Definition des flexiblen Einheitensystems	285
8.6.2	Addition, Subtraktion und Zuweisung	287
8.6.3	Multiplikation und Division	293
8.6.4	Potenz- und Wurzelfunktionen	298
8.6.5	Konvertierung zum SI-Einheitensystem	300
	Literatur	303
9	Speicheroptimierung mit Bitfeldern	305
9.1	Vorbetrachtung	305
9.2	Bitfelder mit Templates realisieren	307
9.2.1	Bitfelder definieren	307
9.2.2	Speichern des Bitfelds	308
9.2.3	Lesen der Elemente	310
9.2.4	Schreiben der Elemente	313
9.2.5	Lesen und Schreiben eines Zeitstempels im Bitfeld	316
9.2.6	Weitere nützliche Methoden für Bitfelder	317
9.3	Umsetzung der Bitfelder mit Typlisten	317
9.3.1	Anlegen der Typliste	318
9.3.2	Definition des Bitfelds	320
9.3.3	Lesen und Schreiben der Elemente	321

9.4	Umsetzung der Bitfelder mit variadischen Templates	324
9.4.1	Definition des Bitfelds	324
9.4.2	Lesen und Schreiben der Elemente	325
9.5	Prüfen von Gleitkommazahlen mit Bitfeldern	327
	Literatur	330
10	Metaprogrammierung mit Gleitkommazahlen	331
10.1	Gleitkommaarithmetik zur Übersetzungszeit	331
10.1.1	Erweiterte mathematische Funktionen	331
10.1.2	Vektor- und Matrizenrechnung	334
10.2	Gleitkommazahlen als Templateargumente	342
10.2.1	Definition und Normalisierung	342
10.2.2	Realisierung der Grundrechenarten	344
10.2.3	Erweiterte mathematische Funktionen	352
10.3	Bruchrechnung mit Templateargumenten	359
10.3.1	Definition und Normalisierung	359
10.3.2	Realisierung der Grundrechenarten	361
10.3.3	Erweiterte mathematische Funktionen	372
	Literatur	376
11	Weitere Beispiele der Metaprogrammierung	377
11.1	Berechnung von Primzahlen	377
11.1.1	Berechnung von kleinen Primzahlen	377
11.1.2	Optimierung des Primzahlentests	379
11.1.3	Berechnung von großen Primzahlen	382
11.2	Berechnung des Osterdatums	385
	Literatur	386
	Stichwortverzeichnis	387

Die Softwaretechnik ist einem ständigen Wandel unterzogen. Um heute effizient und wirtschaftlich Software entwickeln zu können ist es notwendig, einen großen Teil des Quellcodes aus allgemeingültigen Bibliotheken zu benutzen, um den kundenspezifischen Anteil möglichst gering zu halten. Software aus freien und tausendfach bewährten Bibliotheken, wie der Standard Template Library (STL) oder der Boost Bibliothek, sind dabei zu bevorzugen. Bei der Entwicklung von neuer Software sollte die erste Überlegung immer in Richtung einer allgemeingültigen Umsetzung gehen. So entstehen mit der Zeit Softwaresystemfamilien, die flexibel für unterschiedlichste Anwendungen eingesetzt werden können.

Auf die Entwicklung der Softwaresystemfamilien baut die generative Programmierung auf. Sie zielt dabei auf die nahezu automatische Erstellung von Softwaresystemen auf Grundlage der flexiblen Softwaresystemfamilien. Hierbei kommen vor allem Generatoren zum Einsatz, die auf Grundlage von Konfigurationswissen das Softwaresystem erzeugen.

Die Metaprogrammierung lässt sich wiederum als Programmierung des Generators auffassen, deren Ergebnis ein neuer Quellcode ist. Aber auch eine Erweiterung der Programmiersprache um neue Funktionalitäten oder optimierten Quellcode ist mit ihr möglich. Die griechische Vorsilbe *meta* steht dabei für *über* oder *hinter* und deutet auf eine übergeordnete Programmierung hin. Die hier vorgestellte C++-Metaprogrammierung beschreibt die Technik der Metaprogrammierung nur mit Mitteln der Programmiersprache C++ selbst. Hierfür stehen in C++ zwei unterschiedliche Mechanismen zur Verfügung, die Präprozessor- und die Template-Metaprogrammierung.

Die Präprozessor-Metaprogrammierung befasst sich mit dem Generieren und Einsetzen von Codesegmenten in den vorhandenen Quellcode vor dem Kompilieren. Es erfolgt dabei weder eine Typ- noch eine Syntaxprüfung, weshalb der Einsatz des Präprozessors auf diejenigen Einsatzfälle zu beschränken ist, die mit der Template-Metaprogrammierung nicht abgedeckt werden können.