

Skript zur Vorlesung
Effizientes Programmieren
mit C/C++
(für Verwaltungsinformatik)

WS 2022/23

Dipl. - Inf. (FH) Stefan Müller
Hochschule Hof

Dieses Skript einschließlich aller seiner Teile ist urheberrechtlich geschützt und nur für den Gebrauch der Studierenden im Rahmen der Vorlesung "Effizientes Programmieren in C/C++" an der Hochschule Hof bestimmt. Eine Weitergabe an Dritte (auch von Teilen) oder eine Veröffentlichung (auch eine Veröffentlichung im Inter- oder Intranet) ist somit nicht zulässig.

Inhaltsverzeichnis

Inhalt

1	Die Entwicklung von C nach C++	5
1.1	Vom Quellcode zur ausführbaren Datei	8
2	Grundlegende Sprachkonstrukte.....	9
2.1	Die Main-Funktion	9
2.2	Kommentare.....	9
2.3	Variablen, Konstanten und Ausdrücke.....	10
2.4	Bildschirm Ausgaben	10
2.4.1	Escape-Sequenzen.....	11
2.5	Tastatureingaben	11
3	Datentypen, Variablen und Konstanten	13
3.1	Deklaration und Initialisierung	13
3.1.1	Wahrheitswerte	14
3.1.2	Zeichen	14
3.1.3	Integer-Variablen und -Konstanten.....	15
3.1.4	Floating-Point-Variablen und -Konstanten.....	17
3.2	Implizite und explizite Typkonvertierung	18
3.3	Konstanten-Definition	18
3.4	Arrays	20
3.4.1	Array-Deklaration und Initialisierung	20
3.4.2	Mehrdimensionale Arrays	23
4	Operatoren	24
4.1	Grundlagen.....	24
4.2	Übersicht der C++-Operatoren.....	25
5	Kontrollstrukturen	26
5.1	Vorbemerkungen.....	26
5.2	Verzweigungen	27
5.3	Schleifen.....	30
5.4	Sprunganweisungen	31
6	Funktionen und verteilte Entwicklung.....	32
6.1	Funktionsdefinition, -deklaration und -aufruf.....	32
6.2	Verteilte Entwicklung.....	34
6.3	Sichtbarkeit von Variablen und Konstanten	35
6.4	Rekursive Funktionen	38
6.5	Referenzvariablen (Verweise auf andere Variablen)	39
7	Der Präprozessor	41
7.1	Funktionsweise.....	41
7.2	Einbinden von Dateien	41
7.3	Makrodefinitionen	42
7.4	Bedingte Kompilierung	43
7.5	Namensräume (namespaces).....	45
8	Pointer	47
8.1	Grundbegriffe	47
8.2	Zeiger und const	48
8.3	Übergabe von Pointern an Funktionen	49
8.4	Pointer Arithmetik.....	50
8.5	Zusammenhang zwischen Array und Pointer	50
8.6	Spezielle Pointer	52
8.7	Generische Zeiger (void-Pointer)	53
9	C-Strings.....	55
9.1	Deklaration und Initialisierung	55
9.2	Ein- und Ausgabe von C-Strings.....	56
9.3	String-Funktionen in C	58
10	Komplexe Datentypen	60
10.1	Selbstdefinierte Typen.....	60

10.2	Enums.....	60
10.3	Structs.....	62
10.4	Unions.....	65
11	Dynamische Datenstrukturen	67
11.1	Einführung	67
11.2	Speicher reservieren mit malloc und realloc	68
11.3	Häufige Fehler	70
12	Erste C++-Erweiterungen	71
12.1	Kleinere nicht-objektorientierte Erweiterungen.....	71
12.2	new und delete	74
13	Das Klassenkonzept	75
13.1	„Auf zu neuen Welten...“ eine Einführung.....	75
13.1.1	Datenabstraktion (Generalisierung)	76
13.1.2	Datenkapselung	76
13.1.3	Vererbung	76
13.1.4	Polymorphie	77
13.2	Der Konstruktor	78
13.3	Der Copy-Konstruktor	81
13.4	Der Destruktor	82
14	Vererbung	84
14.1	Vererbung und Zugriffsschutz	84
14.2	Vererbung von einer Basisklasse (Einfachvererbung)	84
14.3	Vererbung von mehreren Klassen (Mehrfachvererbung)	87
14.4	Virtuelle Vererbung.....	88
14.5	Abstrakte Klassen (Vorgabe von Schnittstellen)	89
15	Operatorüberladung.....	93
15.1	Vorbetrachtungen	93
15.2	Operatorüberladung am Beispiel der komplexen Zahlen	95
15.3	Die Operatoren << und >>.....	98
16	Templates	101
16.1	Einführung	101
16.2	Generische Funktionen (Funktions-Templates)	101
16.3	Spezialisierung	103
16.3.1	Nach Datentypen optimierte Templates	103
16.3.2	Templates oder überladene Funktionen verwenden?	104
16.4	Generische Klassen (Klassen-Templates)	104
16.4.1	Generische Klassen mit Template Parametern	104
16.4.2	Generische Klassen mit Template und Non-template Parametern	107
16.5	Template Meta-Programmierung	108
16.6	Template Zusammenfassung	109
17	Übergreifende Konzepte und deren Umsetzung	111
17.1	Friend – Funktionen und Klassen.....	111
17.2	Polymorphismus (Implementierung in C++)	113
18	Grafische Anwendungen	116
18.1	Klassenbibliotheken zur Entwicklung grafischer Anwendungen in C++	116
18.2	Die Qt-Klassenbibliothek	117
18.3	Qt-Konzept mit Beispielen	118
18.3.1	Vordefinierte Signale und Slots	118
18.3.2	Beispiel für selbsterstellte Signale und Slots	119
18.4	Model-View-Controller Konzept.....	120

Model/View ist eine Technik, die zur Trennung von Daten und deren Ansicht verwendet wird. Sogenannte "Widgets" arbeiten mit "Datenmengen" (data sets). Standard Widgets sind leider nicht dazu konzipiert Daten von deren Sicht darauf zu trennen. Das ist der Grund, warum Qt 5 zwei unterschiedliche Arten von Widgets besitzt. Beide Arten von Widgets scheinen gleich zu sein, doch Sie interagieren mit den Daten unterschiedlich.....

1 Die Entwicklung von C nach C++

Die Programmiersprache C entstand Anfang der siebziger Jahre im Zusammenhang mit dem Betriebssystem Unix in den AT&T Bell Laboratories. Eine wesentliche Rolle spielte dabei, Ende der siebziger Jahre die Entwicklung des Betriebssystems Unix (Version 7), dessen Kernel von Dennis Ritchie mit der Programmiersprache C anstatt wie bisher üblich in Assembler programmiert wurde. Aus diesen Erfahrungen heraus entstand das 1978 erschienene Buch "Brian W. Kernighan und Dennis M. Ritchie, „The C Programming Language“, Prentice Hall, Englewood Cliffs N.Y. (1978)", das lange Zeit als Quasi-Standard (sog. K&R-Standard) für die Programmiersprache C galt. Die in diesem Buch gemachten Festlegungen bezüglich Syntax und Semantik der Programmiersprache C stellten den kleinsten gemeinsamen Nenner für effektive C-Programme dar.

Erst im Jahre 1988 hat das ANSI-Komitee (*ANSI – American National Standards Institute*) den ANSI-C Sprachstandard für die Programmiersprache C veröffentlicht. C-Compiler sollten einem klar definierten ANSI-Standard entsprechen. Da die Softwarehersteller gerne proprietäre Erweiterungen in ihren Produkten verwenden, um sich und ihr Produkt von anderen Herstellern herauszuheben. Die ANSI-C Kompatibilität kann optional durch Compileroptionen oder ein „Häkchen“ in den Optionseinstellungen der integrierten Entwicklungsumgebungen erreicht werden.

Aufgrund dieses Entstehungsprozesses besitzt die Programmiersprache C folgende Vorteile:

- C ist eine kompakte Sprache. Sie ist eng in der Implementierung an Assembler angelehnt und bietet daher eine hohe Performance der ausführbaren Programme.
- In C geschriebene Programme sind effizient bezüglich ihrer geringen Programmgröße.
- C ist sehr flexibel, d.h. C lässt dem Programmierer einen breiten Spielraum. C wurde für das strukturierte Programmieren konzipiert.
- C ist im Vergleich zu anderen Sprachen sehr gut standardisiert und deshalb sind Quellcode-Dateien auf Zielsysteme, die einen ANSI-C Compiler anbieten, portierbar.
- C-Compiler existieren für alle wesentlichen Rechnerplattformen.
- C ist Bestandteil der MAC-OSx (clang) und Unix-Programmierungsumgebung (gcc/g++)
- C wird häufig im Umfeld von IoT verwendet, steht somit auch auf SBC's zur Verfügung (Raspberry & Co.)

In einer ersten Version wurde C++ „C with Classes“ genannt. Diese Version wurde in den 80er Jahren von Bjarne Stroustrup aus C weiterentwickelt. C++ ist als die nächste Generation der Programmiersprache C, gewissermaßen als eine Obermenge zu verstehen. Dabei wurde C vor allem um das Konzept der Objekt-Orientierung erweitert.

Bjarne Stroustrup begann 1979 mit seiner Arbeit an „C with Classes“, nachdem er bei einer großen Simulationsaufgabe, die in [Simula](#) entwickelt worden war, Probleme hatte.

Er fügte zunächst folgende c++-Sprach-Konstrukte hinzu:

- Klassen
- Konstruktoren und Destruktoren
- Vererbung (noch ohne Polymorphismus)
- Typüberprüfung

Die Implementierung von „C mit Klassen“ bestand in Form von Präprozessor-Anweisungen, die Makros genannt werden. Sie werden durch einen Präprozessor, der beim Umwandlungsprozess **vor** dem Compiler ausgeführt wird, aufgelöst. Der Präprozessor erzeugt aus dem „C mit Klassen“-Sourcecode durch Textersetzung einen Sourcecode in C-Syntax, der dann anschließend fehlerfrei durch den C-Compiler in Objektcode übersetzt werden kann.

Ab 1982 führte Bjarne Stroustrup die Entwicklung von "C mit Klassen" weiter, welche 1990 in einem ersten Standard von C++ (V3.0) mündete. In C++ umfasste in diese Version folgende C++-Erweiterungen:

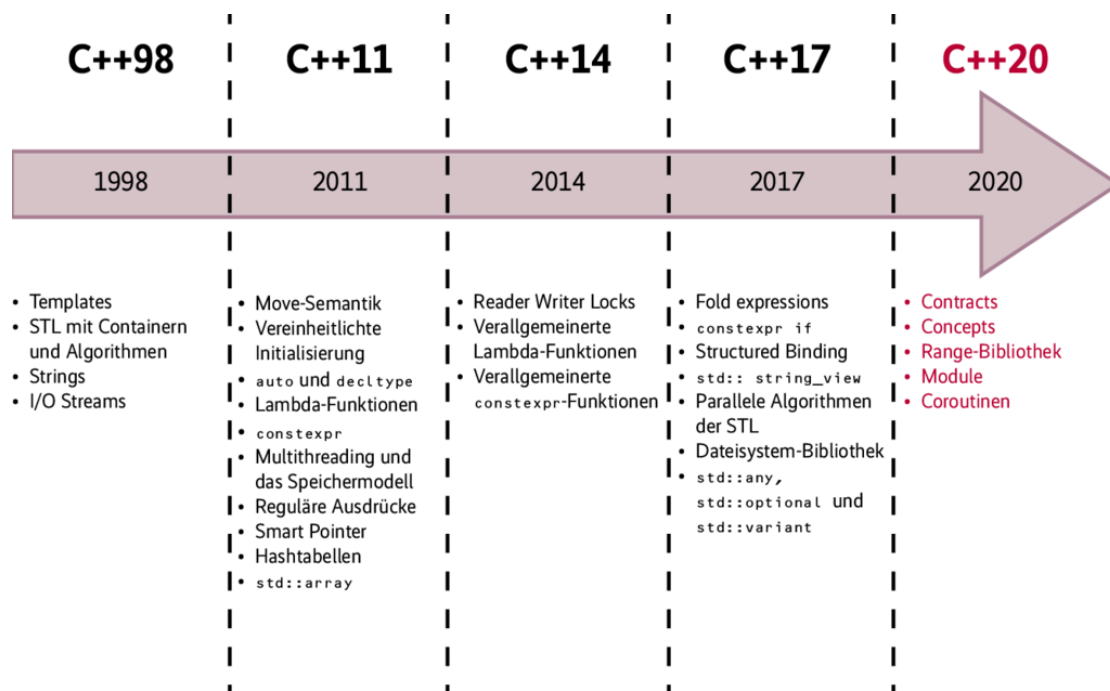
- virtuelle Funktionen
- Überladen von Funktionen und Operatoren
- Referenzen
- Speichermanagement
- Templates

1991 fand das erste Treffen der ISO Workgroup 21 statt. 1998 wurde dann ein "vorläufig endgültiger" Standard von der Arbeitsgruppe verabschiedet. Im wesentlichen wurde noch folgende Funktionalität hinzugefügt:

- Ausnahmebehandlung
- Namensräume
- und ein Typinformationssystem zur Laufzeit des Programms

Dieser ANSI/ISO-Standard ist (mit umfangreichen Erweiterungen) auch heute noch gültig und wird vielseitig eingesetzt. Beispielsweise wurde das Betriebssystem UNIX auf C++ umgeschrieben.

Zum Zeitpunkt im Jahre 2022 hat es bisher folgende Standards gegeben:



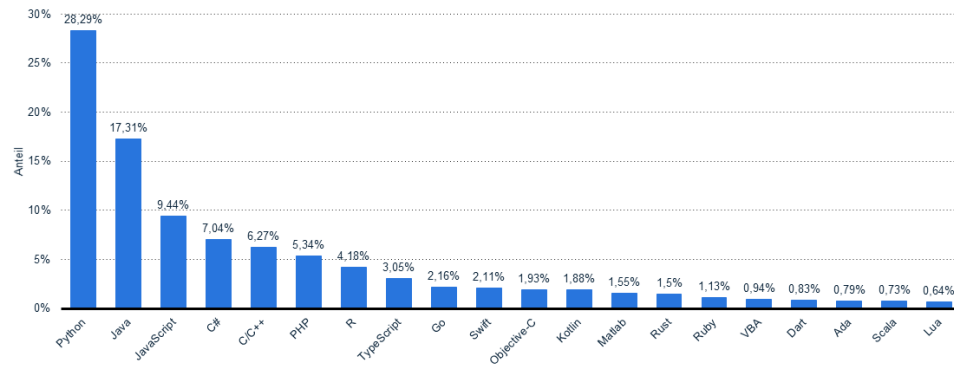
Quelle: Stand 2022, Zur [Heise Magazin Webseite](#)

Bjarne Stroustrup ist für die enge Anlehnung von C++ an C oft kritisiert worden, aber der Erfolg der Programmiersprache C++ gibt ihm im Nachhinein wohl recht.

Nach wie vor ist C/C++ weit verbreitet und eine der gängigsten objektorientierten Programmiersprachen.

Die beliebtesten Programmiersprachen weltweit laut PYPL-Index im September 2022

Beliebteste Programmiersprachen weltweit laut PYPL-Index im September 2022



Hinweis: Weltweit
Weitere Angaben zu dieser Statistik, sowie Erläuterungen zu Fußnoten, sind auf [Seite 8](#) zu finden.
Quelle: PYPL, ID: 678722

statista

Quelle: Statista, Stand 2022 [zur Webseite](#)

Verwendet man C/C++ - so wie wir - als Lehrsprache, so sollten unbedingt die Regeln für strukturiertes Programmieren bekannt sein und auch beachtet werden. Wir sollten also immer im Auge behalten, lesbare, effektive und wartbare Programme (nicht nur funktionierende Programme) zu schreiben.

Da man C primär als Teilmenge von C++ betrachten kann, reduziert sich die Beschreibung von C im Wesentlichen auf die folgenden zwei Aspekte:

- Welche Anteile von C sind keine Teilmenge von C++?
- Welche C++-Erweiterungen fehlen in C und wie werden Sie durch C++-Features ersetzt?

1.1 Vom Quellcode zur ausführbaren Datei

Ein C/C++-Programm wird in folgenden Schritten aus einer Quelle in ein ausführbares Programm umgewandelt:

- Erzeugen eines Quellprogramms in einem Editor
Es wird eine Textdatei erzeugt und editiert. In dieser Textdatei wird das Quellprogramm (*Source Code*) gespeichert. Im Zusammenhang mit der Programmentwicklung in C ist es üblich als Datei-Endung für die Quelldatei ".c" zu wählen. Im Verlauf der Vorlesung werden wir so genannte Header-Dateien kennenlernen. Sie enthalten keine vollständigen Programme sondern nur Programmbausteine und enden üblicherweise auf ".h". Für C++-Programme wählt man üblicherweise die Datei-Endungen *.cpp (Windows) bzw. *.cc (unter Linux). Entsprechend enden auf Header-Dateien auf *.hpp (Windows) oder *.hh (unter Linux).
- Ein Präcompiler nimmt Textersetzungen vor
Bevor der Sourcecode vollständig übersetzt werden kann, werden vordefinierte Schlüsselwörter, die mit dem Zeichen „#“ eingeleitet werden, von einem sog. Präcompiler durch C-Code ersetzt. Das Paradebeispiel hierfür ist die Anweisung `#include <stdio.h>`. Die angegebene Headerdatei wird vom Präprozessor gelesen und direkt in den Quellcode eingefügt.
- Übersetzen des Quellprogramms
Nach der Ergänzung der Quelldatei durch den Präcompiler wird diese von einem Compiler übersetzt. Ein C/C++-Compiler ist ein Programm, das aus der Quelldatei eine Folge von Maschinenbefehlen erzeugt, die vom jeweiligen Prozessor direkt ausgeführt werden können. (Eine Zwischenstufe mit Byte-Code, der auf einer virtuellen Maschine ausgeführt wird, gibt es nicht.) Ein Compiler erzeugt also Binärcode für jeweils nur eine Prozessorfamilie (AMD, ARM oder Intel). Der vom Compiler erzeugte Binärcode wird in einer Datei abgespeichert, welche in der Regel die Extension ".obj" oder ".so" hat. Solch eine "Objekt"-Datei enthält Maschinenbefehle, welche vom einem Texteditor nicht mehr interpretiert werden können. Findet ein Compiler eine Stelle im Quellcode, die nicht den Regeln der Programmiersprache entspricht (Syntaxfehler), so erzeugt er lediglich eine entsprechende Fehlermeldung aber keine „Objekt“-Datei.
- Binden des ausführbaren Programms
Die nach einer fehlerfreien Übersetzung des Compilers erzeugte Objekt-Datei ist noch nicht "lauffähig". Fast jedes Programm verwendet Unterprogramme, z.B. aus der so genannten Standardbibliothek. Der Objektdatei fehlt noch die Information, wo sich die Einsprungadressen der externen Unterprogramme befinden. Ein Linker verknüpft die in unterschiedlichen Dateien liegenden Programmteile zu einem ausführbaren Programm. Für diese Programme ist die Erweiterung ".com" oder ".exe" üblich. Unter Unix ist keine Erweiterung notwendig, da die Information im Header der Datei gespeichert ist. Es werden aus Gründen der Übersichtlichkeit aber manchmal Dateiendungen wie „.out“ oder „.o“ verwendet.

Editor, Präcompiler, Compiler und Linker können völlig eigenständige Programme sein, welche der Programmierer explizit der Reihe nach aufruft. In den letzten Jahren hat sich jedoch das Konzept der Integrierten Entwicklungsumgebung (*IDE – Integrated Development Environment*) durchgesetzt, bei dem alle Aufgaben in einem Programmrahmen integriert zur Verfügung stehen. Dies erleichtert den Bedienungskomfort für den Programmierer. Trotzdem sollte der Programmierer immer wissen in welcher Phase der Umwandlung er sich gerade befindet, um die Gründe für eine Fehlersituation zuordnen zu können.