HANSER

Leseprobe

Norbert Heiderich, Wolfgang Meyer

Technische Probleme lösen mit C/C++

Von der Analyse bis zur Dokumentation

ISBN (Buch): 978-3-446-44784-4

ISBN (E-Book): 978-3-446-44905-3

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-44784-4

sowie im Buchhandel.

Vorwort des Herausgebers

Was können Sie mit diesem Buch lernen?

Wenn Sie mit diesem Lernbuch arbeiten, dann erwerben Sie umfassende Erkenntnisse, die Sie zur Problemlösungsfähigkeit beim Programmieren mit der Hochsprache C/C++ führen.

Der Umfang dessen, was wir Ihnen anbieten, orientiert sich an

- den Studienplänen der Fachhochschulen für technische Studiengänge,
- den Lehrplänen der Fachschulen für Technik,
- den Anforderungen der Programmierpraxis,
- dem Stand der einschlägigen, professionellen Softwareentwicklung.

Sie werden systematisch, schrittweise und an ausgewählten Beispielen mit der Entwicklungsumgebung Visual C++ (VC++) von Microsoft vertraut gemacht.

Dabei gehen Sie folgenden Strukturelementen und Verfahrensweisen nach:

- Wie stellt sich die Entwicklungsumgebung dar?
- Welche grundlegenden Sprach- und Steuerungswerkzeuge gilt es kennenzulernen und an einfachen Beispielen anzuwenden?
- Wie wird ein Problem strukturiert programmiert?
- Wie muss die Software dokumentiert und getestet werden?
- Was meint objektorientierte Programmierung?

Wer kann mit diesem Buch lernen?

Jeder, der

- sich weiterbilden möchte,
- die Grundlagen der elektronischen Datenverarbeitung beherrscht,
- Kenntnisse in den Grundlagen der elementaren Mathematik besitzt,
- bereit ist, sich mit technischen, mathematischen und kommerziellen Fragestellungen auseinanderzusetzen.

Das können sein:

- Studenten an Fachhochschulen und Berufsakademien,
- Studenten an Fachschulen für Technik,
- Schüler an beruflichen Gymnasien und Berufsoberschulen,
- Schüler in der Assistentenausbildung,
- Meister, Facharbeiter und Gesellen während und nach der Ausbildung,
- Umschüler und Rehabilitanden,
- Teilnehmer an Fort- und Weiterbildungskursen,
- Autodidakten.

Wie können Sie mit diesem Buch lernen?

Ganz gleich, ob Sie mit diesem Buch in Hochschule, Schule, Betrieb, Lehrgang oder zu Hause lernen, es wird Ihnen Freude machen!

Warum?

Ganz einfach, **weil wir Ihnen ein Buch empfehlen, das in seiner Gestaltung die Grundgesetze des menschlichen Lernens beachtet.**

– Ein Lernbuch also! –

Sie setzen sich kapitelweise mit den Lehr-, Lerninhalten auseinander. Diese sind in überschaubaren Lernsequenzen schrittweise dargestellt. Die zunächst verbal formulierten Lehr-, Lerninhalte werden danach in die softwarespezifische Darstellung umgesetzt. An ausgewählten Beispielen konkretisiert und veranschaulichen die Autoren diese Lehr- bzw. Lerninhalte.

– Also auch ein unterrichtsbegleitendes Lehr-/Lernbuch mit Beispielen! –

Für das Suchen bestimmter Inhalte steht Ihnen das Inhaltsverzeichnis am Anfang des Buches zur Verfügung. Sachwörter finden Sie am Ende des Buches. Bücher zur vertiefenden und erweiterten Anwendung sind im Literaturverzeichnis zusammengestellt.

– Selbstverständlich mit Sachwortregister, Inhalts-und Literaturverzeichnis! –

Sicherlich werden Sie durch intensives Arbeiten mit diesem Buch Ihre „Bemerkungen zur Sache“ unterbringen und es so zu Ihrem individuellen Arbeitsmittel ausweiten:

– So wird am Ende Ihr Buch entstanden sein! –

Möglich wurde dieses Buch für Sie durch die Bereitschaft der Autoren und die intensive Unterstützung des Verlages mit seinen Mitarbeitern. Ihnen sollten wir herzlich danken.

Beim Lernen wünsche ich Ihnen viel Freude und Erfolg!

Ihr Herausgeber

Manfred Mettke

Inhalt

2 Erste Gehversuche mit C/C++

Dieses Kapitel beschäftigt sich mit einigen grundlegenden Aspekten der Programmiersprache C/C++. Neben der Frage, warum es sinnvoll ist, gerade mit C/C++ zu arbeiten, werden Funktionsweisen der Komponenten der Entwicklungsumgebung betrachtet und erläutert. In den folgenden Kapiteln werden zunächst Beispiele in klassischem C als Konsolenanwendungen realisiert, bevor später objektorientiert mit C++ weitergearbeitet wird. Dann sind die Beispiele auch mit grafischen Oberflächen ausgestattet.

2.1 Warum gerade C/C++?

Wer C/C++ erlernen will, hat sich für eine Programmiersprache entschieden, die auf fast allen Rechnertypen und unter fast allen Betriebssystemen verfügbar ist. Es steht Ihnen, anders als bei vielen anderen Programmiersprachen, auf den verschiedensten Entwicklungsplattformen eine genormte Standard-Bibliothek zur Verfügung. Damit gelingt eine einheitliche Implementierung der mit dieser Programmiersprache erstellten Programme mit sehr hoher Geschwindigkeit.

C wird auch als Highlevel-Assembler bezeichnet, also als Programmiersprache, die sehr nah an der Maschinensprache ist. Dies beruht auf der Tatsache, dass der Kern (bzw. Kernel) aller gängigen Betriebssysteme in C geschrieben wurde. Damit eignet sich C/C++ auch in besonderem Maße für die Systemprogrammierung, also für Programme, die für den Betrieb von Rechenanlagen erforderlich sind.

Dank der relativ einfachen Struktur und dem geringen Umfang der eigentlichen Sprache, d. h. der verfügbaren Schlüsselworte der Programmiersprache, war es möglich, **C-Compiler**, also spezielle Programme zur Übersetzung des vom Programmierer erstellten Codes in eine maschinenverständliche Sprache, für alle Arten von Prozessorplattformen zu entwickeln, so dass die Programmiersprache C/C++ heute für die gesamte Leistungspalette vom Mikrocontroller bis zu High-End-Rechnern verfügbar ist. Für den Entwickler von Software bedeutet dies: Egal für welche Prozessorplattform programmiert wird, einen C-Compiler wird man für das relevante Zielsystem bekommen. Man braucht sich nicht um eine Programmierung zu kümmern, die spezifisch für den jeweiligen Zielprozessor ist. In den meis-

ten Fällen wird es möglich sein, die auf einer Plattform entwickelte Anwendung auf einer anderen Plattform auszuführen. Der erforderliche Anpassungsaufwand ist in aller Regel sehr überschaubar.

Das bedeutet nicht, dass man fertige Programme von einer Plattform auf eine andere übertragen kann (etwa von einem Windows-PC auf einen Linux-PC) und diese dann auf der neuen Plattform (also unter Linux) sofort wieder funktionieren. Vielmehr ist nur die problemlose Übertragung der Quelltexte auf ein neues System gemeint, auf dem diese dann mit dem entsprechenden Compiler und Linker (ein Linker oder Binder ist ein Programm, das einzelne Programmmodule zu einem ausführbaren Programm verbindet) in ein funktionierendes Programm umzuwandeln sind!

Die Tatsache, dass Programme, die in C/C++ geschrieben werden, sehr klein sind (nur in Assembler – also Maschinensprache – geschriebene Programme sind noch kleiner), macht C/C++ zu einer wichtigen Programmiersprache im Bereich Embedded Systems (also Systemen, die stark einschränkenden Randbedingungen unterliegen, wie geringe Kosten, Platz-, Energie- und Speicherverbrauch) und der Mikrocontroller-Programmierung, wo Speicherplatz ebenfalls sehr kostbar ist.

Ein C/C++-Programm wird mithilfe eines **Compilers** (dem Übersetzer des Quelltextes) aus einer oder mehreren einfachen Textdateien zu Objektcodedateien übersetzt. Diese Objektcodedateien werden anschließend von einem **Linker** (bzw. Linkage-Editor = Binder, Werkzeug für das Zusammenfügen übersetzter Programmteile) mit den erforderlichen Systembibliotheken zu einer ausführbaren Datei (der Executable – oder kurz EXE-Datei) zusammengebunden.

Jedes ausführbare C/C++-Programm besitzt eine Hauptfunktion. In C wird diese Hauptfunktion als `main` bezeichnet.

Damit das Betriebssystem erkennen kann, wo der Einstiegspunkt für den Ablauf eines C/C++-Programms zu finden ist, muss diese Namenskonvention unbedingt eingehalten werden. Auch wenn andere Entwicklungsumgebungen als das Visual Studio von Microsoft oder andere Compiler eingesetzt werden, ändert sich an diesem Einstiegspunkt nichts. Variieren kann allenfalls die Parametrisierung (also die Art, Anzahl oder der Datentyp der Übergabeparameter) dieser Hauptfunktion. Dieser Aspekt wird später in Kapitel 4.8, in dem es um Funktionen gehen wird, noch ausführlich erläutert.

Darüber hinaus ist es natürlich auch möglich, eigene Programme und/oder Funktionen in eigenen Bibliotheken zusammenzufassen, um diese später erneut benutzen zu können. Diese Bibliotheken können bei einem späteren Bindevorgang durch den Linker wieder verwendet werden, damit diese dann zu einem neuen Programm hinzugebunden werden.

2.2 Compiler und Interpreter

Die höheren Programmiersprachen (dazu zählen u. a. FORTRAN, ALGOL und C/C++) sind entwickelt worden, damit die Beschreibung eines Lösungsverfahrens, der Algorithmus, in einer für Menschen einfacher lesbaren Form erfolgen kann und nicht in Maschinencode vorliegen muss. Der Programmierer als der Anwender solcher Programmiersprachen soll sich also auf die Lösung seiner konkreten Aufgabenstellung konzentrieren können, ohne sich zu sehr mit den Interna des Rechners beschäftigen zu müssen. Ein weiterer, nicht zu vernachlässigender Vorteil ist, dass Programmiersprachen normalerweise unabhängig von der Maschine sind, auf der die Programme ausgeführt werden.

Die Tatsache, dass es zwei verschiedene Verfahren zur Erzeugung von Programmen gibt, deutet das Problem an, das sich hinter der Verwendung einer Programmiersprache verbirgt: die Programme können nicht mehr direkt und unmittelbar vom Computer gelesen und zur Ausführung gebracht werden, sondern sie müssen erst in eine vom Computer interpretierbare passende Darstellungsform gebracht werden.

Eine Möglichkeit Quellprogramme (also die vom Programmierer erstellten und für den Programmierer lesbaren Textdateien mit den Programmen) zu übersetzen, sind **Interpreter**. Bei ihnen wird das Programm Zeile für Zeile gelesen und bewertet, wobei Schreibfehler oder Verstöße gegen die Regeln der Programmiersprache, Syntaxfehler, festgestellt werden. Danach führt der Interpreter die mit den Anweisungen verbundenen Befehle und Aktionen aus. Die Arbeitsweise solcher Interpreter kann wie in Bild 2.1 dargestellt werden.

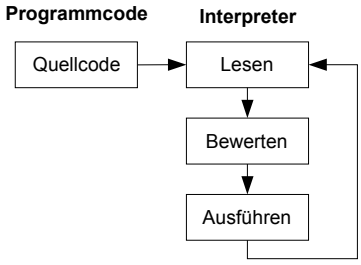

Bild 2.1
Arbeitsweise eines Interpreters

Der große Vorteil interpretierter Programme besteht darin, dass es möglich ist, schnell einmal etwas auszuprobieren oder während des Programmablaufs einzugreifen und Werte von Variablen zu betrachten oder zu verändern. Für professionelle Projekte dagegen sind Interpreter eher ungeeignet. Zuerst einmal ist da das Problem der mangelnden Geschwindigkeit zu nennen. Denn egal, ob ein Programm zum ersten oder tausendsten Mal ausgeführt wird, alle zugehörigen Programmzeilen durchlaufen immer wieder den Zyklus: *Lesen – Bewerten – Ausführen.*

Außerdem ist nur ein geringer Schutz des Quellcodes vor Eingriffen von außen gegeben. Der Anwender hat jederzeit die Möglichkeit den Code zu manipulieren, ohne sich mit dem Programmierer abzusprechen. Es entstehen so in der Praxis ziemlich schnell alternative Programmversionen, die bei einem neuen Release zu großen Problemen führen.

Der andere Lösungsansatz, um ein ausführbares Programm zu erzeugen, führt über den Einsatz eines Compilers. Ein **Compiler** ist ein Programm, das einen Quellcode einliest und

ihn zunächst auf syntaktische Fehler untersucht. Werden keine wirklichen Fehler festgestellt, so wird der Quellcode in eine maschinenlesbare Form übersetzt.

Nach dem abschließenden Schritt des Bindens (Linken) liegt das Programm in einer Form vor, die auf dem Rechner, auf dem es verarbeitet wurde, lauffähig (also ausführbar) ist (engl. executable, daher die Extension (Programmendung) EXE). Dieses Programm kann nun unabhängig von Quellcode und Compiler ausgeführt werden. Diese Arbeitsweise lässt sich wie in Bild 2.2 darstellen.

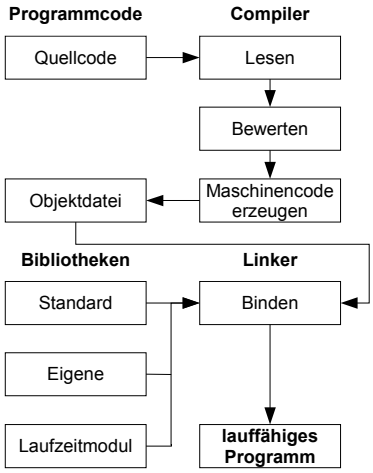

Bild 2.2
Arbeitsweise eines Compilers und Linkers

Der große Vorteil dieser Vorgehensweise liegt darin, dass die Programme ohne Preisgabe des Quellcodes lauffähig sind. So können z. B. auch im Quellcode verwendete Betriebsgeheimnisse von keinem Anwender mehr eingesehen werden. Vor allen Dingen aber laufen solche Programme sehr viel schneller ab als interpretierte Programme, da ja die Interpretation jedes Programmstatements (also jeder Programmzeile) entfällt. Die Überprüfung hat bereits bei der Kompilierung (also der Übersetzung in die Object-Datei) stattgefunden. Ein gewisser Nachteil besteht zweifellos darin, dass nun bei jeder Programmänderung der komplette Zyklus *Editieren – Übersetzen – Starten* durchlaufen werden muss, bevor man etwas in einem Programm ausprobieren kann. Das kostet in der Testphase natürlich etwas mehr Zeit als bei interpretativen Programmen. Denn auch beim Auftreten eines syntaktischen Fehlers muss man immer erst wieder das Programm in den Editor laden, den Fehler beheben und die Übersetzung (Kompilierung) von neuem starten.

Deutlich entschärft wird dieses Problem dadurch, dass die meisten Compiler heutzutage mit einer sogenannten integrierten Entwicklungsumgebung ausgestattet sind, in der der Editor und der Compiler eine Einheit bilden. Damit verkürzen sich die Bearbeitungsschritte erheblich.

2.3 Übersetzen eines C/C++-Programms

Nachdem der Quellcode eingegeben ist, muss dieser in eine für den Computer verständliche Sprache übersetzt werden. An dieser Stelle wird ein **Object** erzeugt. Dieser Prozess stellt einen Zwischenschritt bei der Herstellung eines ausführbaren Programms dar. Das Werkzeug, mit dem hier gearbeitet wird, ist ein Compiler. Das ist ein eigenständiges Programm zur Übersetzung des Quellcodes in ein Object.

1. Der erste Schritt ist der **Präprozessorlauf.** Bei dem Programm, das hier zum Einsatz kommt, handelt es sich um einen Textersetzer, der spezielle Dateien, die Headerdateien, in den Programmtext importiert. Zusätzlich vermag es der Präprozessor, Makros und symbolische Konstanten in die ihnen zugeordneten Anweisungsfolgen und Werte zu übersetzen und bestimmte Programmteile (Kommentare) vor der Übersetzung auszublenden. Vom Präprozessor wird eine temporäre Zwischendatei erzeugt, die aber nach dem Übersetzungsvorgang wieder zerstört wird und somit keinen dauerhaften zusätzlichen Speicherplatz auf der Festplatte beansprucht.
2. Im zweiten Schritt testet der Compiler, ob die Schreibweisen und der Aufbau des Programms den Regeln der Programmiersprache entsprechen. Hier spricht man von der **syntaktischen Analyse.** Ein dabei auftretender Fehler wird dementsprechend Syntaxfehler genannt. Moderne Compiler sammeln nun die Fehler und brechen die Übersetzung erst bei der Überschreitung einer gewissen Fehlerobergrenze ab. Anschließend kann der Anwender von der Fehlerliste aus zu den Programmstellen springen, die der Compiler moniert hat. Hier gilt es zu bedenken, dass es sich nicht immer nur um eine einzige Stelle handeln muss, an der der Fehler liegen kann. Evtl. ist schon einige Zeilen vorher ein Fehler programmiert worden, der erst jetzt zu einem Syntaxproblem führt und den Compiler zur Meldung veranlasst. Der Programmierer korrigiert die beanstandeten Zeilen und startet den Übersetzungslauf erneut. Je nach Komfort oder Aussagekraft der Fehlermeldungen ist die Analyse der Fehlerquellen aufwendig. In jedem Fall bedarf es einiger Übung, um aus den teilweise sehr kryptischen Fehlermeldungen die Fehlerquelle zu erkennen.
3. Nach einigen Durchgängen ist das Programm in der Regel frei von Syntaxfehlern. Der Compiler hat dann den **Objectcode** erzeugt. Dieser ist prinzipiell schon maschinenlesbar, kann allerdings noch nicht ausgeführt werden, da einerseits weitere Objectdateien evtl. hinzuzufügen sind, andererseits aber immer Bibliotheksdateien und das Laufzeitmodul hinzugebunden werden müssen.
4. Das **ausführbare Programm**, das dann auf der Plattform, auf der die Entwicklung durchgeführt wurde, lauffähig ist – und in aller Regel auch nur auf dieser Plattform, denn C/C++ ist eine plattformabhängige Programmiersprache –, erzeugt letztendlich der Linker. Hier werden nun alle Bestandteile, die zur Lauffähigkeit des Programms benötigt werden, also das Laufzeitmodul, die Bibliotheksfunktionen und Objectdateien, zu einer ausführbaren Programmdatei zusammengeführt (gebunden). Auch in diesem Schritt besteht noch die Möglichkeit, dass der Programmierer Fehler begeht. So sind Fehler, die in diesem Schritt entdeckt werden z. B. fehlende Funktionsdefinitionen, fehlende externe Variablen oder doppelte Funktionsdefinitionen. Auch in solchen Fällen muss der Programmierer wieder zurück bis in den Quellcode, um die Fehler zu korrigieren. Anschließend sind die aufgeführten Schritte erneut alle zu durchlaufen.

2.4 Programmstart

Nachdem der Quellcode fehlerfrei übersetzt und gebunden wurde, liegt eine Datei vor, die ein ausführbares Programm-Modul darstellt. Sie ist gekennzeichnet durch die **Extension** (Dateierweiterung) EXE. Dieses Programm kann nun, wie jedes andere Programm auch, gestartet werden. An dieser Stelle beginnt standardmäßig die Testphase, in der der Programmierer versucht herauszubekommen, ob die von ihm realisierte Lösung immer (d. h. mit unterschiedlichen Eingabedaten) zu einem vernünftigen und erwarteten Ergebnis führt. Dazu gehören selbstverständlich auch die Auswahl von geeigneten Testdaten sowie die Dokumentation von erwarteten und erzielten Resultaten.

Selten werden die Programme auf Anhieb die an sie gestellten Aufgaben mit den erwarteten Ergebnissen liefern. Es ist also nun wieder der Quellcode heranzuziehen, um ihn so zu berichtigen, dass die Ergebnisse den Erwartungen entsprechen.

Die häufigsten Fehler, die in diesem Zusammenhang gemacht werden, sind:

▪ Designfehler:	Die Lösung ist nicht angemessen. Zur Realisierung des Programms sind zu viele Annahmen getroffen worden, die nicht in jedem (Beispiel-) Fall tatsächlich auch vorliegen, oder es ist eine falsche Schrittfolge der einzelnen durchzuführenden Arbeitsschritte innerhalb des Quellcodes umgesetzt worden.
▪ Logische Fehler:	Z. B. falsche Abfragebedingungen oder Reihenfolge der Programmschritte falsch angeordnet.
▪ Sprachbedingte Fehler:	Da C/C++ eine äußerst knapp gehaltene Sprache ist, können Laufzeitfehler daraus resultieren, dass man sich schlicht vertippt hat. Es kann also passieren, dass man einen syntaktisch korrekten Programmteil erstellt, der aber doch ein logisch falsches Konstrukt enthält, das der Compiler allerdings nicht identifizieren kann. Man spricht hier von semantischen Fehlern.

Durch die sequentielle Arbeitsweise (Korrektur von Fehlern – Test der neuen Version – Korrektur von Fehlern – Test der neuen Version …) entstehen immer wieder neue Versionen des Programms, die man durch entsprechende Vergabe von Programmnamen oder Versionsnummern dokumentieren kann und sollte. Ist dann irgendwann der endgültige Programmstand erreicht, können diese Zwischenstände wieder entsorgt werden.

Index

D

E

F

G

T

U

V

W

Z