Inhaltsverzeichnis

Vorwort

„Schon wieder ein neues Programmierbuch“ mag der eine oder andere stöhnen, der in Werbung oder Buchhandel auf dieses Buch stößt. Meine Antwort darauf ist „ja und zwar aus gutem Grund“, denn es ist durchaus lohnend, sich mit (Free-) Pascal und Lazarus näher zu befassen!

Was spricht für das Arbeiten mit (Free-) Pascal und Lazarus?

Für den Einsteiger bietet Lazarus eine komfortable Entwicklungsumgebung für die Erstellung von Programmen mit grafischer Bedienoberfläche. Lazarus basiert auf der Programmiersprache Pascal bzw. deren Implementation Free Pascal. Pascal wurde nicht zuletzt für Ausbildungszwecke entwickelt und hat aufgrund seiner sauberen Definition seine Eignung für die für die Schul- und Hochschulausbildung unter Beweis gestellt.

Für Umsteiger von Delphi und anderen Programmierumgebungen ist Lazarus vor allem dann interessant, wenn man bei der Gestaltung seiner Applikation bisweilen gewisse Kompromisse eingehen kann. Dass Lazarus Delphi momentan, was den Umfang angeht, noch nicht ebenbürtig ist, kann man nicht wegdiskutieren. Man kann jedoch annehmen, dass die stetig wachsende Gemeinschaft der Lazarus-Nutzer und vor allem auch der aktiven Kontributoren daran arbeiten wird, diese Lücke zu schließen.

Lazarus wächst rasch und kontinuierlich

Gerade aufgrund der nach Ansicht des Verfassers derzeit sehr restriktiven Lizenzpolitik von Embarcadero – dem derzeitigen Anbieter von Delphi - bietet die kostenlose Verfügbarkeit von Lazarus nicht nur Schulen und interessierten Privatleute sondern auch Unternehmen die Möglichkeit, Kosten im Bereich der Lizenzierung zu reduzieren und gleichzeitig in einer weitgehend Delphi-kompatiblen Programmierumgebung zu arbeiten. Lazarus ist als Multiplattformumgebung angelegt, d. h. dass damit Programme für verschiedene Betriebssystemumgebungen entwickelt werden können. Der Schwerpunkt dieses Buches liegt allerdings beim Einsatz unter Windows.

Lazarus ist eine Multiplattformumgebung. Der Buchschwerpunkt liegt auf aber Windows-Anwendungen

Dieses Buch wendet sich sowohl an Ein- als an Umsteiger. Dabei folgt es einem klaren didaktischen Konzept. Anhand typischer Beispielaufgaben werden Schritt für Schritt Lernziele festgelegt um das Know-How weiter zu entwickeln. Innerhalb einer jeden Aufgabe wird dargelegt, welche Eigenschaften von Pascal und Lazarus für die Bearbeitung der Aufgabe von Bedeutung sind und wie man sie einsetzen sollte. Schließlich wird die komplette Lösung der jeweiligen Aufgabe vorgestellt. Abgeschlossen wird jeder Lernschritt durch Übungsaufgaben, mittels derer das im jeweiligen Kapitel erworbene Wissen vertieft werden kann.

Das klare didaktische Konzept unterstützt **Ein**- und **Um**steiger.

Dabei werden u. a. folgende Themen behandelt:

- Programmierung in Pascal von den Anfängen bis zu den fortgeschrittenen Feinheiten (Zeiger, Rekursion, Dateiverkehr …)
- Implementation zahlreicher grundlegender Algorithmen.

- Programmierung von Aufgaben aus dem Mathematikunterricht der Realschulen, der Gymnasien und der Anfangssemester der Hochschulen.
- Pascalentwicklung mit Verwendung einer grafischen Benutzeroberfläche
- Elementare Grafikprogrammierung

Nach dem Studium dieses Buches sind Sie in der Lage ein repräsentatives Spektrum von Aufgaben zu lösen, die durch die Sprachdefinition von (Free-) Pascal abgedeckt sind. Anders als wenn Sie sich ausschließlich auf Free-Pascal stützen würden, werden Ihre Programme fast von Anfang an mit einer professionellen grafischen Bedienoberfläche versehen sein, die sich auf die Softwarebibliothek LCL (Lazarus Component Library) stützt. LCL ist auf Sprachebene (Pascal) mit der von Delphi verwendeten VCL (Visual Component Libray) weitgehend schnittstellenkompatibel, sodass Delphi- und Lazarus-Know-How in den meisten Fällen austauschbar wird. Der zweite Teil (Band 6 dieser Reihe „Informatik ganz einfach“) wird weiteren Themen wie Datenbanken, Internet und speziellen Fragen der Systemprogrammierung gewidmet sein.

Die grafische Bedienoberfläche gehört selbstverständlich dazu.

Frühzeitig wird in diesem Buch auf die Erstellung zeitgemäßer Bedienoberflächen Wert gelegt. Die für die Erstellung der grafischen Bedienoberfläche erforderlichen Kenntnisse in objektorientierter Programmierung werden anfänglich pragmatisch entsprechend dem aktuellen Bedarf vermittelt (Kapitel 9). Später erfolgt dann eine systematische Einführung (Kapitel 15).

Aktuelle Informationen zu diesem Buch, wie Korrekturen, Antworten auf Leserfragen oder auch Bestellmöglichkeiten für CDs mit dem Programmcode zu meinen Büchern finden Sie im Internet unter **www.okomedien.de**

Ein Buch wie dieses kann nicht ohne die tatkräftige Unterstützung Dritter zustande kommen. Ich danke meinen Studenten für die zahlreichen Diskussionen und Anregungen, meinem Sohn Matthias für das Korrekturlesen und die Erprobung der Programmbeispiele und nicht zuletzt meiner Frau Ruth für ihre Geduld.

Aktuelle Informationen zum Thema des Buches erhalten Sie unter der Service-Adresse **www.okomedien.de/service/laz1**. Ihre Zugangsdaten erhalten Sie gegen Übersendung (Email) des Kaufbelegs und Nennung Ihrer Email-Adresse.

Sie brennen sicher schon darauf, Ihre Arbeit mit Lazarus zu beginnen. Laden Sie die Integrierte Entwicklungsumgebung Lazarus aus dem Internet herunter. Installieren und starten Sie Lazarus (näheres finden Sie in 2.3.1) und arbeiten Sie sich dann Schritt für Schritt anhand des Lehrtexts und der Beispiele vorwärts.

Ich wünsche Ihnen eine erfolgreiche Einarbeitung in Lazarus und anschließend eine erfolgreiche Projektarbeit!

Oberkochen im Juni 2016

Wilfried Koch

1. Hintergrund und Anwendung dieses Buches

1.1. Zielsetzung

Dieses Buch führt anhand leicht verständlicher Beispiele aus der Schul- und Hochschulausbildung in die Programmierung mit einer auf Pascal basierenden Sprache ein. Es wendet sich einerseits an komplette Programmieranfänger, bietet aber auch Fortgeschrittenen und Umsteigern von anderen Entwicklungsumgebungen die erforderlichen Informationen.

1.2. Programmieren – worum geht es dabei überhaupt?

Ein Programm ist eine Arbeitsvorschrift für einen Rechner. Es gibt – wenigstens im Falle von FreePascal-/Lazarus-Programmen an, welche Arbeitsschritte ein Rechner zur Bearbeitung einer Aufgabe ausführen muss.

Die Erstellung eines Programms ist letztlich die Erstellung einer Arbeitsvorschrift. Arbeitsvorschriften sind Ihnen zur Genüge bekannt: Kochrezepte, Gebrauchsanweisungen, Bastelanweisungen aber auch Musiknoten gehören beispielsweise dazu. Während in den hier genannten Fällen Menschen die Arbeitsvorschriften ausführen, sind es im Falle von Programmen Rechner.

1.2.1. Programmiersprachen

Menschen können nach entsprechender Ausbildung Arbeitsvorschriften, die ihnen in natürlichen Sprachen wie Deutsch, Englisch oder auch Chinesisch aber auch in grafischer Form (z. B. Musiknoten) erteilt werden, korrekt interpretieren. Hingegen „verstehen“ die meisten Rechner bis heute nur Programmiersprachen.

Wenn man Programmiersprachen betrachtet so muss man zwischen den maschinennahen (niederen) und den höheren Programmiersprachen unterscheiden.

Programmierung in Maschinensprache:

Zeitaufwand und Fehleranfälligkeit sind hoch.

Zu den niederen Programmiersprachen gehören vor allem die Maschinensprachen der jeweiligen Rechner. Sie sind im allgemeinen prozessorspezifisch[1]. Sie sind für den Rechner für den sie entwickelt wurden direkt und problemlos, für den Menschen aber vergleichsweise schwer verständlich. Sie bestehen letztlich aus ziemlich abstrakten Zahlenfolgen. Für Menschen ist es zwar möglich in einer Maschinensprache zu programmieren, die Arbeitsgeschwindigkeit bei der Programmerstellung ist bei dieser Vorgehensweise jedoch unakzeptabel niedrig und die Fehleranfällig-

1 Der Prozessor ist derjenige Teil Ihres Rechners, der die eigentliche „Rechenarbeit“ leistet.

keit in der Regel sehr hoch. Aus diesem Grund wurde schon früh Möglichkeiten gesucht, um die Programmierarbeit wirtschaftlicher gestalteten. Man „erfand“ die Programmiersprachen.

In einem ersten Schritt entstanden die noch sehr maschinennahen Assemblersprachen. Üblicherweise wird bei diesen Sprachen der Operationscode des Befehls durch einen mnemotechnischen Buchstabencode (z. B. ADD statt 100_{16} für Addition). Operanden können wahlweise durch Zahlen oder alphanumerische Symbole dargestellt werden.

Aus verschiedenen Gründen und vor verschiedenen fachlichen und geschäftspolitischen Hintergründen entstanden in den letzten 50 Jahren zahlreiche höhere Programmiersprachen. Die meisten von ihnen basieren auf einer Art mathematischer Notation die mit ein paar englischen Vokabeln kombiniert ist. Der folgende Text zeigt einen Ausschnitt aus einem Programm in der höheren Programmiersprache Pascal:

```
if a >= b then
  c := 1
else
  c := 2;
```

Dieses kurze Programmstück sagt aus, dass falls der Wert von `a` größer oder gleich dem Wert von `b` ist, der Variablen `c` der Wert `1` zugewiesen wird. Andernfalls bekommt `c` den Wert `2`.

1.2.2. Syntax und Semantik

Als Syntax bezeichnet man die Lehre vom Satzbau. So wie in einer natürlichen Sprache aus dem verfügbaren Wortschatz nach bestimmten Regeln korrekte Sätze gebildet werden, werden auch die Sätze (Audrücke, Anweisungen...) eines Programms nach festen Regeln gebildet. Die Syntaxregeln einer Programmiersprache können mit Methoden wie der Backus-Naur-Form (BNF) oder Syntaxdiagrammen wie sie in diesem Buch vorzugsweise zur Anwendung kommen dargestellt werden.

Die Semantik beschäftigt sich mit der Bedeutung der Sätze. Syntaktisch korrekte Sätze können semantisch falsch sein. Diese semantischen Fehler können zu einem gewissen Teil vom Compiler entdeckt werden. Sie dürfen nicht mit logischen Fehlern verwechselt werden.

Beispiel:

```
var a, b, c: integer;
….....
….....
a := b or c;
```

Der Programmausschnitt ist zwar syntaktisch korrekt, aber semantisch fehlerhaft, da mit `or` nur Daten vom Typ `boolean` verknüpft werden dürfen.

In natürlicher Sprache ist das vergleichbar mit den Sätzen

- Herr Maier isst sein Auto.

und

- Herr Maier wäscht sein Auto.

Beide Sätze sind syntaktisch korrekt. Semantisch korrekt ist nur der letztere.

1.2.3. Vom Compiler zur Integrierten Entwicklungsumgebung

Die Texte, die der Programmierer[2] in dieser Sprache erstellt, kann der Rechner nicht direkt verstehen. Sie müssen für ihn in seine Maschinensprache übersetzt werden. Diese Übersetzung kann vor oder während der Ausführung des Programms erfolgen.

Compilierung vor der Programmausführung, **Interpretation** während der Programmausführung.

Erfolgt die Übersetzung vor der Programmausführung, so spricht man von einer Compilierung, das Programm, das die Übersetzung ausführt ist der Compiler.

Die Übersetzung während der Programmausführung bezeichnet man als Interpretation und das dafür eingesetzte Programm entsprechend als Interpreter.

In der Softwareindustrie werden heute sowohl interpretierbare Sprachen als auch compilierbare Sprachen eingesetzt. Compilierbare Sprachen erzielen wesentlich besseres Laufzeitverhalten als interpretierbare. Bei gleicher Aufgabenstellung und gleichem Rechner sind die Laufzeiten von compilierten Programmen kürzer als von interpretierten, da die zeitaufwändige Übersetzung bereits vor der Ausführung erfolgte.

Quellcode→Objektcode (Compiler)

Objektcode→Ausführbares Programm (Linker)

Meist besteht ein Programm aus mehreren Teilen (Modulen). Die Module, die der Programmierer in der (höheren) Programmiersprache - z. B. Pascal - schreibt, nennt man Quellmodule. Werden sie übersetzt, so entsteht daraus die Objektmodule. Die Verknüpfung der Objektmodule durch den Binder (Linker) führt dann zum lauffähigen Programm (Laufzeitprogramm, Laufzeitsystem, Executable).

Editor, Compiler, Binder und Debugger werden heute üblicherweise innerhalb eines einzigen Programms betrieben. Dieses wird als integrierte Entwicklungsumgebung

2 Alles was dieses Buch über den Umgang mit dem Entwicklungssystem Lazarus darstellt gilt selbstverständlich gleichermaßen für männliche und weibliche Nutzer. Der übersichtlicheren Darstellung wegen wird ab sofort nur noch die männliche Form verwendet.

(IEU, engl. Integrated Development Environment (IDE)) bezeichnet. Lazarus ist ein Beispiel für eine solche Integrierte Entwicklungsumgebung. Weitere IEUs sind z. B. Microsoft Visual Studio, Embarcadero RADS, Qt oder Java Swing.

1.3. Pascal – Ein kurzer historischer Überblick

1.3.1. Der Anfang

Pascal ist eine Weiterentwicklung von Algol 60 einer besonders in Europa populären frühen höheren Programmiersprache. Die Sprache Pascal wurde von Niklaus Wirth an der Eidgenössischen Technischen Hochschule in Zürich vor allem in Hinblick auf die Verbesserung der Programmierausbildung entwickelt.

Pascal zwingt zu einem klaren Umgang mit Daten- und Programmstrukturen und fördert auf diese Weise die Erstellung von wohl strukturiertem und gut verständlichem Code

1.3.2. Wichtige Implementationen

Sehr große Verbreitung in der professionellen Programmierung fand Pascal als Borland/Turbo Pascal (später Delphi). Dabei handelt es sich um gegenüber dem Ur-Pascal wesentlich erweiterte und verbesserte Versionen. Diese bieten neben dem ursprünglichen Sprachumfang von Pascal einen einfachen Zugriff auf wichtige Funktionen des Betriebssystems. Später kamen dann noch objektorientierte Elemente hinzu.

1.3.3. Free Pascal und Lazarus

Free Pascal ist ein unter der GNU Lesser General Public License stehendes Open-Source-Projekt, das sich zum Ziel gesetzt hat, einen freien 32 bzw. 64-Bit-Compiler zu erstellen, der 100-prozentig kompatibel zu Turbo Pascal und Delphi sein soll und mittlerweile eine leichte Portierung von Pascal-Programmen auf fast alle gängigen Betriebssysteme und Hardwareplattformen ermöglicht.

Lazarus ist eine Integrierte Entwicklungsumgebung (IEU, engl. IDE für Integrated Development Environment) für Free Pascal, die auch verschiedene Komponenten zur Verfügung stellt. Die IDE ist sehr Delphi-ähnlich gestaltet und verwendet unter Unix das GTK+[3] als Grafik-Toolkit, unter Windows (Win32/Win64/Windows CE)

3 GTK+ = **GTK+** (*GIMP Toolkit*) ist eine freie Komponentenbibliothek unter der LGPL (Lesser Gnu Public License), mit der grafische Benutzeroberflächen (GUI) für Software erstellt werden können.

setzt es auf dem nativen[4] API[5] auf, und auf Apple-Betriebssystemen kann wahlweise das native Carbon-API oder das X-Window-System verwendet werden. Darüber hinaus unterstützt Lazarus Cross Compiling, so dass auch Software für weitere Plattformen wie Windows CE, OS/2, Palm OS oder Nintendo DS entwickelt werden kann. Lazarus ist noch nicht fertiggestellt. In kurzen Abständen werden aber neue Betaversionen veröffentlicht, die – je nach Plattform in unterschiedlichem Maße – einen wachsenden Teil der geplanten Funktionalität implementieren. [LAZHOME]

1.4. Didaktische Ausrichtung

Grundsätzlich ist dieses Buch als Programmierlehrbuch nach dem Prinzip „Learning by Doing“ angelegt. Das Erlernen der Programmiersprache erfolgt dabei durch Erarbeiten von Programmlösungen. Seine Gliederung orientiert sich an den zu erwerbenden Programmierkenntnissen. Diese werden von Kapitel zu Kapitel durch auf einander aufbauende Programmieraufgaben erweitert. D. h. von Aufgabe zu Aufgabe werden die Kenntnisse vertieft. Gleichzeitig dringen Sie von Kapitel zu Kapitel tiefer in die Möglichkeiten der Entwicklungsumgebung Lazarus ein.

Die Kapitel dieses Lehrbuchs sind durchgängig in folgende vier Abschnitte gegliedert:

- Aufgabenstellung: Welche Aufgabe soll das Programm erfüllen? Wie soll es eingesetzt werden?
- Beitrag von Pascal: In diesem Abschnitt wird dargestellt, welche (Free-) Pascal-Sprachelemente Sie zur Lösung der Aufgabe benötigen und was Sie bei ihrer Benutzung beachten müssen. Sowohl für Pascal als auch für Lazarus (nächster Absatz) werden im allgemeinen nur solche Elemente aufgeführt die in vorangegangenen Kapiteln noch nicht besprochen wurden
- Beitrag von Lazarus: Hier erfahren Sie, welche Teile der Entwicklungsumgebung Lazarus Sie zur Programmentwicklung einsetzen müssen und welche Hilfe Sie von diesen erwarten können.
- Lösung: Vorstellung und Erläuterung des kompletten Programmtexts.
- Weitere Aufgaben, ggf. mit Lösungen.

---

4 Nativ von engl. native (=eingeboren) bezeichnet eine Entwicklung oder ein Subsystem, die/das vom Grundsatz her zu einer bestimmten Arbeitsumgebung (Hardware und/oder betriebssystem) gehört.

5 API: Application Programming Interface, Schnittstelle eines Programms A (häufig so wie auch hier eines Betriebssystems), die es ermöglicht die Fähigkeiten dieses Programms A durch ein Programm B zu nutzen.

Wer ein Lehrbuch wie dieses schreibt, steht manchmal vor dem Problem von Ei und Henne, d. h., dass zur Erklärung des momentan erarbeiteten Stoffs idealerweise Elemente erforderlich wären, die erst in späteren Kapiteln erarbeitet werden. Um Sprünge im Text zu vermeiden wird in diesem Fall folgendermaßen vorgegangen:

- Die gestellte Aufgabe wird zunächst mit den bis zum jeweiligen Kapitel vorgestellten verfügbaren Sprachmitteln gelöst. Das führt zu verständlichen, funktionierenden, aber nicht optimalen und vielleicht etwas umständlichen Lösungen.
- Wenn in späteren Kapiteln zusätzliches Wissen erarbeitet wurde, das elegantere Lösungen ermöglicht, wird die Aufgabenstellung nochmals aufgenommen. Ein Beispiel dafür ist der Richtigkeitsnachweis für die DeMorganschen Regeln, der erstmals in 4.3.2 auftritt und für den dann in 11.4.1.1 eine wesentlich elegantere Lösung gezeigt wird.

1.5. Wenn Sie nicht Seite für Seite vorwärts arbeiten wollen.....

Dieses Buch wendet sich sowohl an absolute Anfänger als auch an ein breites Spektrum von Quereinsteigern.

Anfängern ohne Vorkenntnisse empfehle ich, dieses Buch von Anfang an Seite für Seite gründlich durchzuarbeiten.

Wenn Sie schon Pascal-Grundkenntnisse besitzen und vor allem an einer Einarbeitung in die Windows-Programmierung interessiert sind, werden Sie nach einem kurzen Studium von Kapitel 1 wahrscheinlich vor allem die Kapitel 9, 15 bis 18 intensiv durcharbeiten und die übrigen eher zum Nachschlagen nutzen.

Wer sich speziell für die Grafikprogrammierung interessiert wird sich vor allem mit den Kapiteln 16 und 17 befassen.

Wenn Sie eine Lösungshilfe für ein bestimmtes Anwendungsgebiet suchen, dann sollten Sie im Stichwortverzeichnis unter **Anwendungsthemen** nachsehen.