Informationstechnisches Gymnasium Leutkirch

Grundlagen der Programmentwicklung

und Programmierung

Informationstechnik (IT)

Gemäß Bildungsplan für das berufliche Gymnasium der dreijährigen Aufbauform

an der Geschwister-Scholl-Schule Leutkirch

Autor: Peter Wiech

Dipl.-Ing. (FH), Studienrat, MBA

Version: 1.05

Leutkirch, den 07. Oktober 2015

Einleitung

Das vorliegende Unterrichtsmaterial stellt den Einstieg in das Fach Informationstechnik am Informationstechnischen Gymnasium dar. Es beschreibt die Lehrplaneinheit 1 „Grundlagen der Programmentwicklung und Programmierung“ für die Eingangsklasse am beruflichen Gymnasium der dreijährigen Aufbauform und folgt damit dem Amtsblatt des Ministeriums für Kultus, Jugend und Sport Baden-Württemberg (Ausgabe C vom 17. 05. 2004), Heft 10 Angewandte Informationstechnik.

Der Aufbau dieses Unterrichtsmaterials ist wie folgt gegliedert:

Einleitung
Theoretische Abhandlungen
Beispiele und Veranschaulichungen aus der Praxis
Übungen und Musterlösungen
Literaturverzeichnis

Meiner Frau danke ich sehr herzlich für die hilfreiche Unterstützung beim Erstellen dieser Arbeit.

Inhaltsverzeichnis

1 Grundbegriffe und Programmiersprachen

1.1 Einige Grundbegriffe

1.1.1 Programm

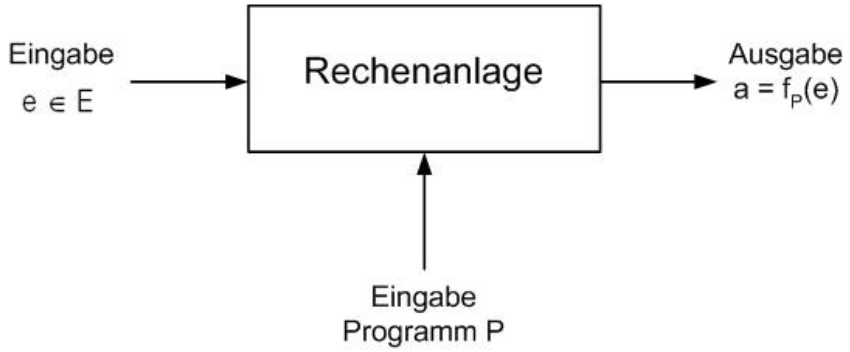
Ein **Programm**[1] ist ein im Voraus festgesetzter Ablauf.

Ein **Programm** ist eine eindeutige, formalisierte Beschreibung von Algorithmen und Datenstrukturen, die durch einen automatisierten Übersetzungsprozess auf einem Computer ablauffähig ist.
Den zur Formulierung eines Programms verwendeten Beschriebungsformalismus bezeichnen wir als **Programmiersprache**.

In der Daten- und Kommunikationstechnik versteht man unter einem Programm eine Folge von Befehlen, die dem Rechner die Vorgehensweise bei der Abarbeitung der Aufgabe mitteilen. Das Programm wird seriell abgearbeitet.
Programme sind problem- oder aufgabenabhängig festgelegte Arbeitsanweisung für Rechenanlagen und sind im exakt definierten und eindeutigen Formalismus einer Programmiersprache verfasst.

Definition aus dem Duden: Syntaktisch korrektes Element einer Programmiersprache.

[1] franz.: programme = schriftliche Bekanntmachung, Tagesordnung, aus dem griech.: prógramma = Vorgeschriebenes, Vorschrift



Mathematisch gesprochen realisiert ein Programm P eine Funktion f_P von der Menge der Eingabedaten E in die Menge der Ausgabedaten A, d.h. f_P: E → A. Zur Ausführung von P auf einer Rechenanlage gibt man P zusammen mit einer Eingabe e ∈ E ein, worauf der Rechner eine Ausgabe a ∈ A erzeugt, für die dann $f_P(e) = a$ gilt.

1.1.2 Algorithmus[2]

Ein Algorithmus ist eine endliche Menge von genau beschriebenen Aneisungen, die unter Benutzung von vorgegebenen Anfangsdaten in einer genau festgelegten Reihenfolge auszuführen sind, um die Lösung eines Problems in endlich vielen Schritten zu ermitteln.

Bei dem Begriff „Algorithmus“ denkt man heute sofort an „Prgrammierung“. Das war nicht immer so. In der Tat gab es Algorithmen schon lange, bevor man auch nur entfernt an Programmierung dachte. Bereits im antiken Griechenland wurden Algorithmen zur Lösung mathematischer Probleme formuliert.

[2] Dieser Begriff geht zurück auf einen Bibliothekar des Kalifen von Bagdad (Abu Jafar Muhammad Ibn Musa Al-Khwarizmi), der um 825 ein Rechenbuch verfasste und dessen Namen in der lateinischen Übersetzung von 1200 als „Algorithmus“ angegeben wurde.

1.1.3 Datenstruktur

Eine Datenstruktur ist ein Modell, das die zur Lösung eines Problems benötigten Informationen (Ausgabedaten, Zwischenergebnisse, Endergebnisse) enthält und für alle Informationen genau festgelegte Zugriffswege bereitstellt.

1.2 Informationsdarstellung

Ein Computer (Rechner, Rechenanlage, Digitalrechner, Datenverarbeitsanlage) ist ein universell einsetzbares, programmgesteuertes Gerät zur automatischen Verarbeitung von Daten. Ein Computer nimmt Eingabewerte (Zahlen, Wörter, elektrische Impulse usw.) entgegen und wandelt sie nach bestimmten Regeln in Ausgabewerte um.

Ein Computer kann aufgrund seines physikalischen Aufbaus nur Informationen in Form von binärer Daten (Null oder Eins) verarbeiten.

Binär bedeutet die Information wird mit zwei unterscheidbaren physikalischen Zuständen dargestellt:

• hohe oder niedrige Spannung auf einer Leitung

hohe Spannung (high Pegel) z.B. 5 V → logische 1

niedrige Spannung (low Pegel) z.B. 0 V → logische 0

• elektrische Ladung / keine Ladung in einer Leitung

elektrische Ladung vorhanden → logische 1

keine elektrische Ladung vorhanden → logische 0

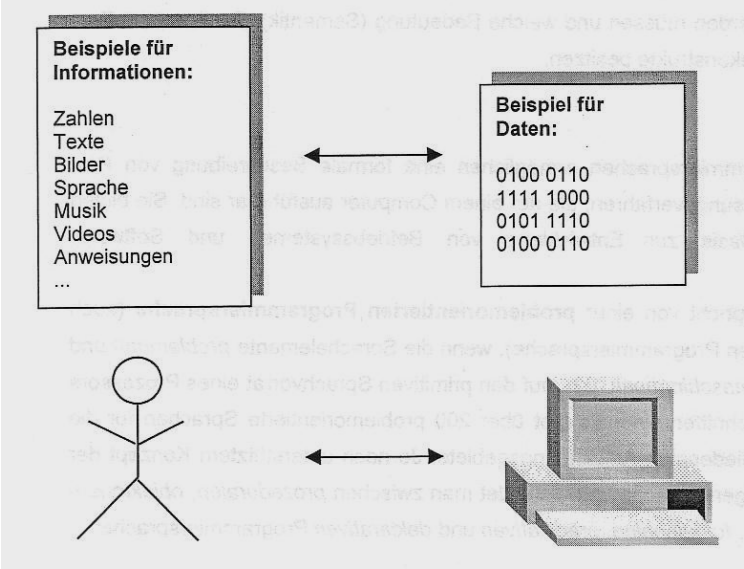
Definition: Daten sind nichts anderes als Bitfolgen.

8 Bit z.B. (01100101) sind 1 Byte

4 Bit z.B. (1001) sind 1 Nibble

Textzeichen werden mit Hilfe des ASCII-Codes in entsprechende Bitfolgen überführt.

z.B. Textzeichen: A → (0100 0001)



1.3 Programmiersprache

Programmiersprachen bilden die wichtigste Schnittstelle zwischen Benutzer und Computer. Jedes Problem, das von einem Computer bearbeitet werden soll, muss zuvor in einer Programmiersprache formuliert werden.

1.3.1 Maschinennahe Programmiersprachen

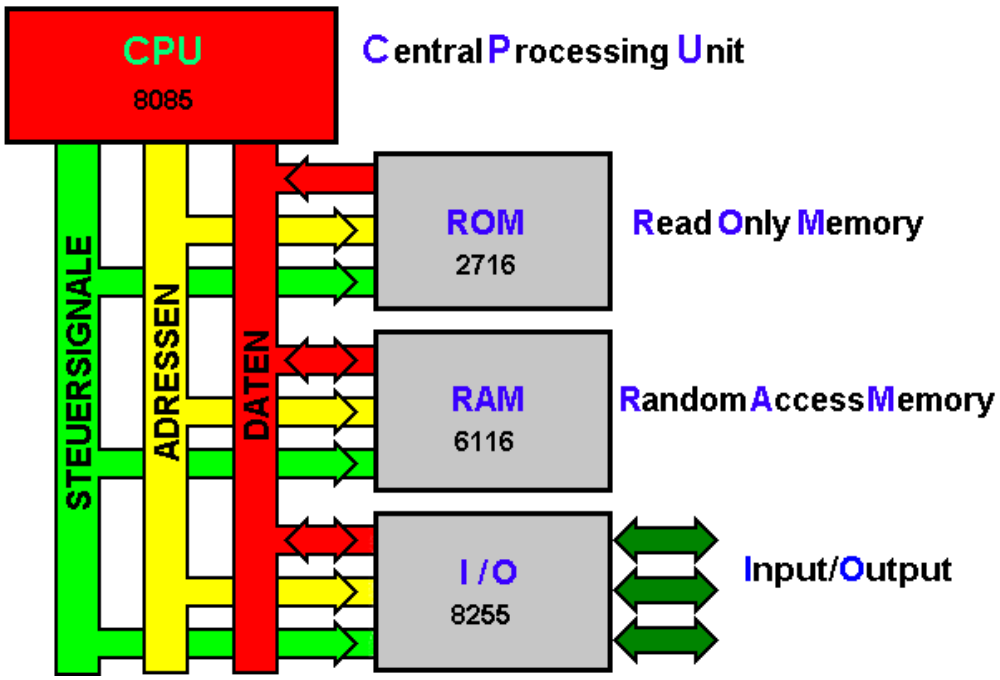
Maschinenprogramm: Programm, welches in einer Maschinensprache formuliert ist. Maschinenprogramme sind die einzigen Programme, die von einem Computer direkt

ausgeführt werden können. Programme in anderen Programmiersprachen müssen entweder interpretiert oder in ein Maschinenprogramm übersetzt werden. Maschinenprogramme sind für Menschen kaum verständlich, da sie aus einer Folge von elementaren, durch Nullen und Einsen dargestellten Befehlen bestehen (Binärcode). Daher schreibt man Maschinenprogramme im Assemblierer[3], der dazu weitere Hilfen bietet.

Maschinensprache: Programmiersprache eines Computers. Programme in dieser Sprache sind im Binärcode dargestellt, wodurch sie vom Prozessor des Computers direkt ausgeführt werden können. Diese Binärdarstellung ist für Menschen kaum lesbar.

Maschinensprache kann nur auf dem Prozessor ablaufen, für den Sie konzipiert ist.

Man spricht auch vom sogenannten Microcode.



[3] Engl. to assemble (sich) versammeln, the assembler der Monteur

Bsp.:
```
3E 5F 3A

2B 4E 2C

1D 5D 7A
```

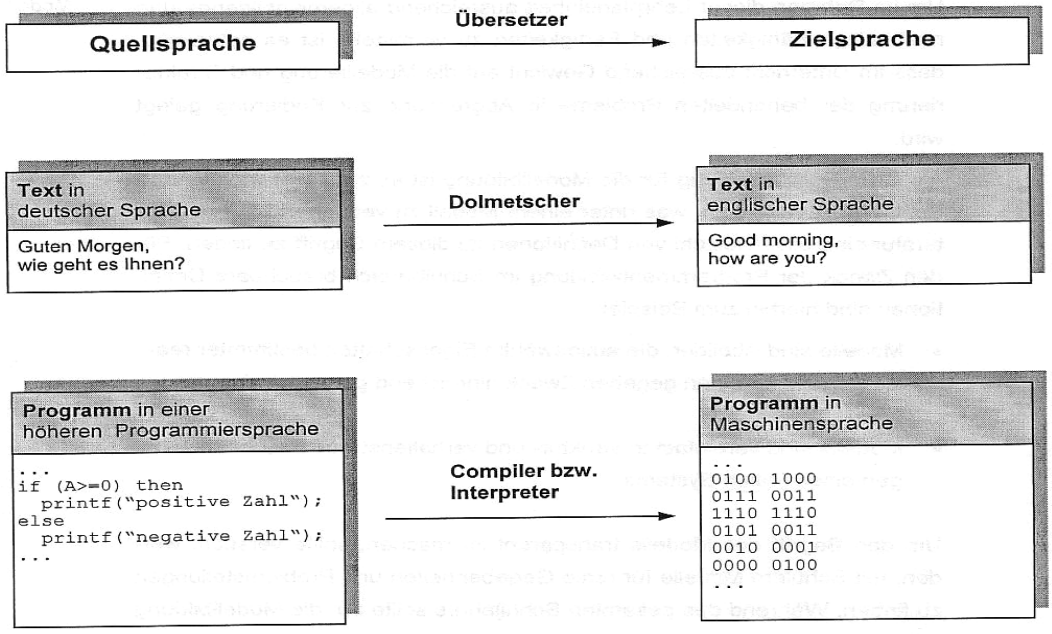
Assemblersprache:

Assembler unterscheidet sich von Maschinensprachen dadurch, dass Befehle durch Befehlswörter (Mnemonics) ausgedrückt werden und dass für die Befehlsparameter symbolische Bezeichner verwendet werden können.

Bsp.:
```
mov ax, 02   (Intel 80x86 assembler)

add ax, 03

mov[0], ax

nop

int 03
```

1.3.2 Höhere Programmiersprachen

Bei höheren Programmiersprachen beinhaltet ein Befehl mehrere Anweisungen in Maschinencode. Befehle einer Hochsprache werden mittels **Compiler** in Maschinencode umgesetzt.



Im wesentlichen unterscheidet man bei den Hochsprachen zwischen prozeduralen– und objektorientierten Programmiersprachen.

1.3.2.1 Prozedurale Programmiersprachen

Programme in prozeduralen Sprachen beschreiben einen Lösungsweg, indem sie exakt festlegen, wie die Lösung zu einem Problem maschinell zu berechnen ist. Es ist unmittelbar ersichtlich, welche Größen gegeben und welche gesucht sind und auf welchem Weg man die gesuchten aus den gegebenen Größen gewinnen kann.

Beispiele prozeduraler Programmiersprachen sind C, Basic, Cobol oder Pascal.

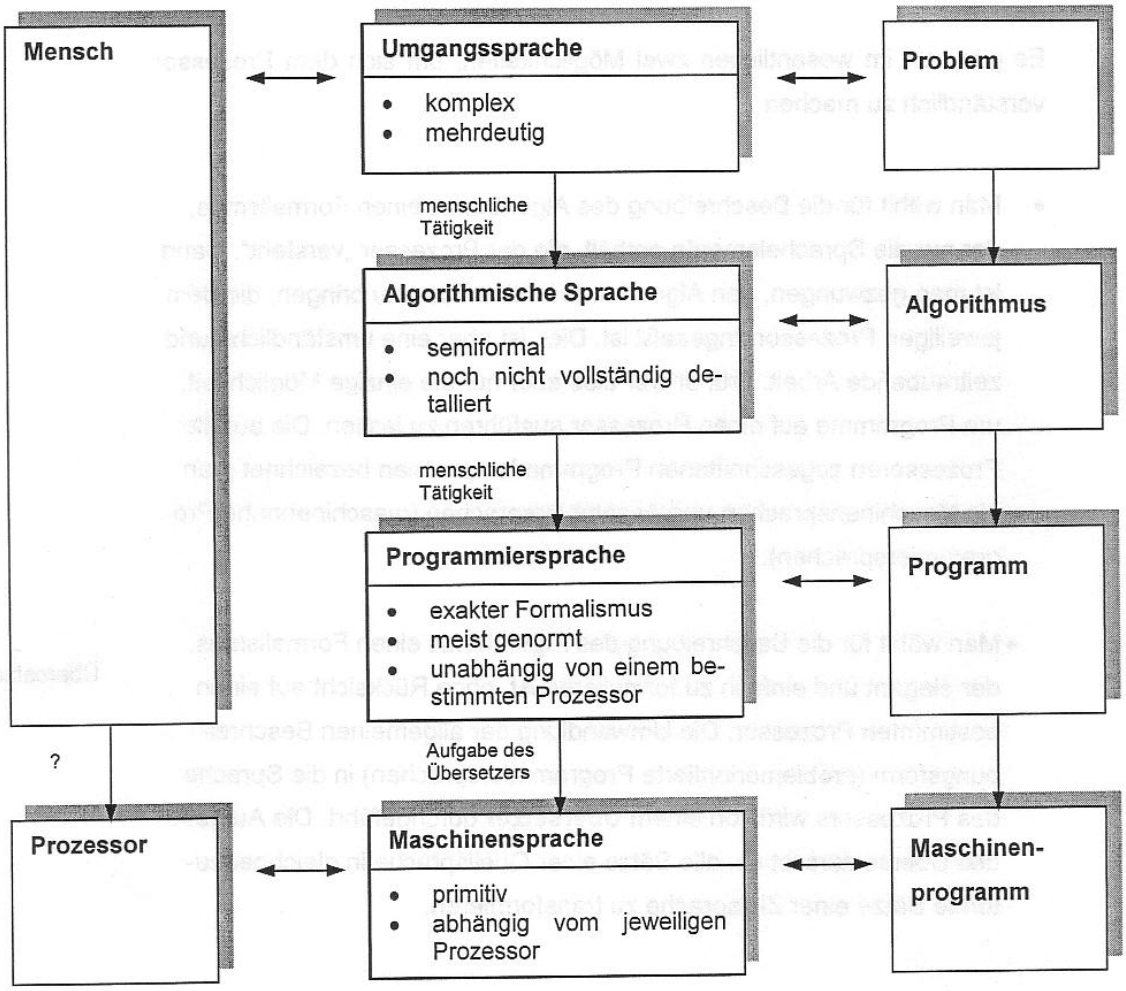
1.3.2.2 Objektorientierte Programmiersprachen (OOP)

Eine Objektorientierte Programmiersprache ist eine Programmiersprache, deren allgemeine Organisationsstruktur die Klasse ist und die damit die objektorientierte Programmierung besonders unterstützt.

Eine Klasse beschreibt das Verhalten einer Gruppe gleichartiger Objekte. Beispielsweise kann man Obst als eine Klasse ansehen, weil man alle Objekte in Obst und Nicht-Obst einteilen kann. Hat man einen konkreten Apfel und eine konkreten Birne auf einem Teller vor sich, dann sind dies Objekte (Instanzen). Man kann von einer Menge gleichartiger Objekte sagen, dass sie zu ein- und derselben Klasse gehören.

Beispiele objektorientierter Programmiersprachen sind C++, SMALLTALK-80, Eiffel und Java.

1.4 Vom Problem zum Maschinenprogramm



1.5 Übungen

1) In welcher Form kann ein Computer Daten verarbeiten?

2) Erklären Sie die Begriffe „Bit“, „Byte“, „Nibble“.

3) Auf welche Art werden Textzeichen im Rechner verarbeitet?

(Welcher Code wird hierfür verwendet?)

4) Nennen Sie Vor- und Nachteile maschinennaher Sprachen.

5) In welchen Bereichen wird heute noch Assembler verwendet?

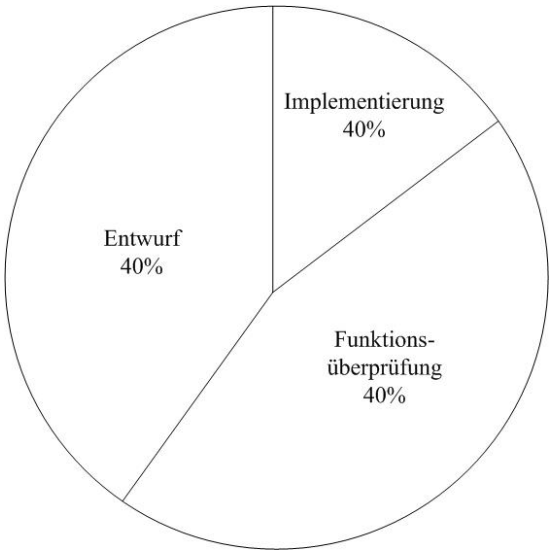
(Bitte mit kurzer Begründung)

6) Nennen Sie zwei prozedurale, sowie zwei objektorientierte Programmiersprachen.

2 Phasen der Programmentwicklung

Programmentwicklung (Programmiertechnik, Softwaretechnologie) wird auch häufig als Software-Engineering bezeichnet. Software-Engineering ist die Anwendung von Prinzipien, Methoden und Techniken auf die Planung (Problemanalyse), den Entwurf, die Implementierung und die Wartung (Funktions- und Leistungsüberprüfung) von Programmen. Der Begriff Software-Engineering steht für die Auffassung, dass die Erstellung, Anpassung und Pflege von Programmen kein „künstlerischer", sondern vorwiegend ein ingenieurmäßig ablaufender Prozess ist. Der Begriff Software-Engineering entstand in den 1960-er Jahren, als die Entwicklung immer größerer Softwaresysteme zunehmend problematischer wurde (Softwarekrise).

Die Qualität eines Programms wird hierbei bestimmt durch seine Benutzerfreundlichkeit, Zuverlässigkeit, Wartbarkeit und durch seine Anpassbarkeit.



Software-Engineering: relativer Zeitbedarf der Phasen Entwurf (und Problemanalyse), Implementierung und Funktionsüberprüfung.

2.1 Problemanalyse

Ziel der Problemanalyse ist es, das zu lösende Problem und alle wichtigen Umgebungsbedingungen vollständig und eindeutig zu erfassen und die Durchführbarkeit der geplanten Software-Entwicklung zu untersuchen. In dieser Phase findet keine konkrete Realisierung statt. Die Problemanalyse gliedert sich in vier Unterphasen, die zum Teil zeitlich nebeneinander ablaufen können:

- Istanalyse
- Sollkonzeptentwicklung
- Durchführbarkeitsstudie
- Projektplanung

Aufgabe: Führe eine Recherche zu den vier Unterphasen durch. Beschreibe kurz deren Bedeutung.

2.2 Entwurf (mit Modellbildung)

In der Entwurfsphase entwickeln die Programmierer ein Modell (Modellbildung) des Gestamtsystems, das, umgesetzt in ein Programm, die Anforderungen erfüllt. Hierzu wird das komplexe Gesamtsystem in unabhängig voneinander realisierbare und in ihrem Zusammenwirken überschaubare Einzelbausteine (Module) unterteilt. Es werden die Funktionen und die Schnittstellen dieser Module beschrieben.

Das Ziel der Modularisierung ist es, den Entwurf transparent und nachvollziehbar zu machen. Ein verbreitetes Konzept, das die obigen Bedingungen erfüllt, ist die hierarchische Modularisierung, zu deren Durchführung es im Wesentlichen zwei

Strategien gibt: die Top-Down-Methode[4] (schrittweise Verfeinerung) und die Bottom-Up-Methode[5].

Grundsätzlich ist die Top-Down-Methode der Bottom-Up-Methode vorzuziehen.

2.3 Implementierung

Erstellung eines lauffähigen Programms, das in seinem Ein-/Ausgabeverhalten der Spezifikation entspricht. Besonders wichtig ist dabei die Wahl der Programmiersprache.

Diese Phase beginnt meist mit der Entscheidung zur Realisierung. Man legt die Datenstrukturen, die grundlegenden Algorithmen und die notwendigen Bibliotheksprogramme fest und klärt, was neu zu programmieren oder wieder zu verwenden ist. Hierbei stehen die einzelnen Module, welche in der Entwurfsphase definiert wurden im Vordergrund.

2.4 Funktions- und Leistungsüberprüfung

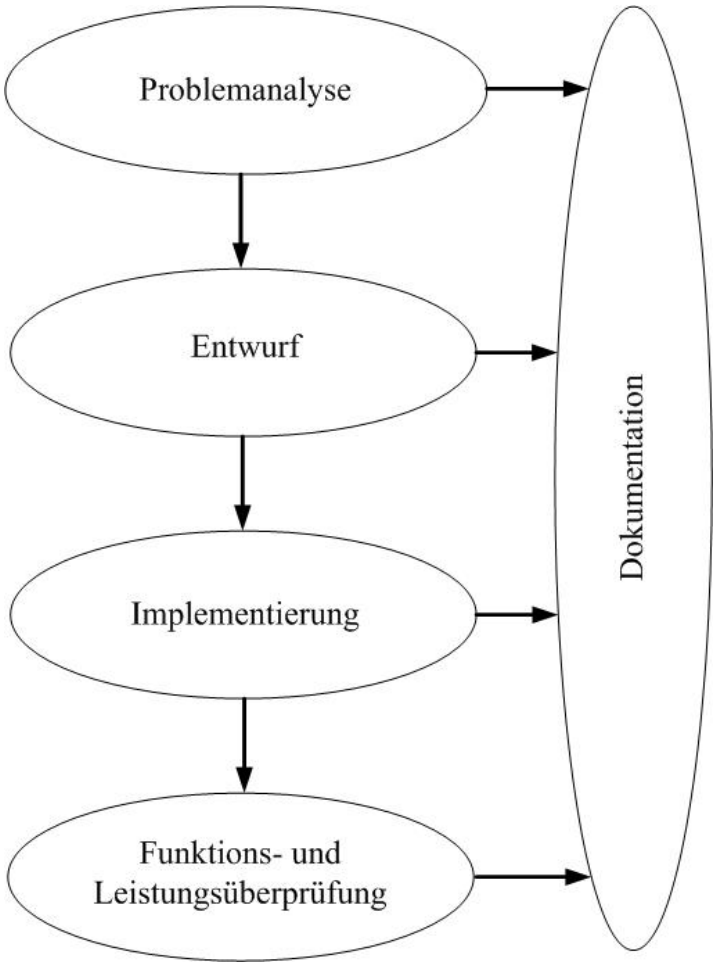
Überprüfung des Ein-/Ausgabeverhaltens von Programmen anhand ihrer Spezifikation. Durch das Testen (Verifikation) von Programmen kann immer nur die Anwesenheit von Fehlern, nie jedoch deren Abwesenheit bewiesen werden, d.h. arbeitet ein Programm für die überprüften Eingaben korrekt, so sagt das nichts darüber aus, ob es für alle Eingaben korrekt arbeitet.

[4] engl. top down „von oben nach unten“

[5] engl. bottom up „von unten nach oben“

2.5 Dokumentation

Verlauf und Ergebnisse aller Phasen müssen **projektbegleitend** dokumentiert werden. Alle diese Dokumente zusammen bilden die Dokumentation. Jedes einzelne Dokument sollte an eine bestimmte Lesergruppe gerichtet sein: **Benutzerdokumentation**, **Entwicklungsdokumentation** und **technische Dokumentation**. Sogenannte CASE-Tools[6] können für größere Softwareentwicklungen eingesetzt werden.



[6] engl. CASE Abk. für computer aided software engineering

3 Entwicklungsumgebung

3.1 A, B, C - die Geschichte von C und C++

Die Programmiersprache C wurde in den Jahren 1972/73 von den beiden C-Erfindern Brian W. Kernighan und Dennis M. Ritchie entwickelt. (Homepage von Dennis M. Ritchie http://www.cs.bell-labs.com/who/dmr/index.html). Sie ist eine Programmiersprache mit Eigenschaften und Sprachelementen aus assemblerähnlichen und höheren Programmiersprachen. Die Bedeutung von C liegt in der Verbreitung des Betriebssystems UNIX, das auf der Sprache C basiert. Die Sprache C zeichnet sich durch eine weitgehende maschinenunabhängige Programmierung aus und ihr Sprachumfang ist vergleichsweise gering.

Die Erweiterung hat C im Jahre 1986 durch objektorientierte Ergänzungen erfahren. Die neue Sprache wurde vom Erfinder Bjarne Stroustrup C++ genannt (Homepage von Bjarne Stroustrup http://www.research.att.com/~bs/). In der Praxis ist die Sprache weit verbreitet; so wurden Teile der Betriebssysteme Microsoft Windows und Unix in C++ verfasst (++ ist der Erhöhungs- oder Inkrementoperator von C, daher resultieren die beiden Pluszeichen).

In die Entwicklung von C++ sind Elemente von Simula67 mit eingeflossen. Einen direkten Nachfolger von C++ gibt es nicht und wird es wohl auch nie geben. Zwei Programmiersprachen sind jedoch stark von C++ inspiriert: Java und C#. Beide sind rein objektorientiert und werden auf dem ausführenden Rechner interpretiert, d.h. es muss dort eine entsprechende Laufzeitumgebung installiert sein. Für Java - Programme die JRE (Java Runtime Environment) und für C# - Programme das .NET-Framework.

3.2 Editor, Compiler, Linker und Debugger

Bevor wir mit der Entwicklungsumgebung Borland C++ arbeiten können, müssen wir noch ein paar Begriffe definieren. Um zu einem lauffähigen Programm zu kommen, muss der Algorithmus zunächst in Programmcode umgesetzt werden. Für das „Schreiben“ dieses Programmcodes benötigen, wie beim Schreiben eines Textes, einen Editor.

3.2.1 Editor

Im Editor wird der **Quelltext** eines C++-Programms erfasst. Dieser Quelltext ist zunächst einmal nichts anderes, als ein reiner ASCII-Text[7] der aus Buchstaben, Zahlen und Sonderzeichen besteht, die jeweils eine Länge von 8 Bit haben. Wir speichern unseren Quelltext mit der Dateinamenserweiterung „.cpp“ und übergeben ihn dem Compiler.

3.2.2 Compiler

Nachdem wir das C++-Programm im Editor erfasst und gespeichert haben, können wir den Quellcode in Maschinencode umwandeln. Dieser Vorgan wird vom Compiler erledigt und wird Kompilieren genannt.

Der Quellcode wird zunächst auf syntaktische[8] Richtigkeit hin geprüft. In dieser Phase erhält man Fehlermeldungen, falls der Compiler auf Probleme wie falsch geschriebene Befehle, fehlende Klammern und Satzzeichen oder unverträgliche Datentypen stößt. Dann werden alle eventuell vorhandenen Kommentare entfernt und Präprozessor-Direktiven aufgelöst.

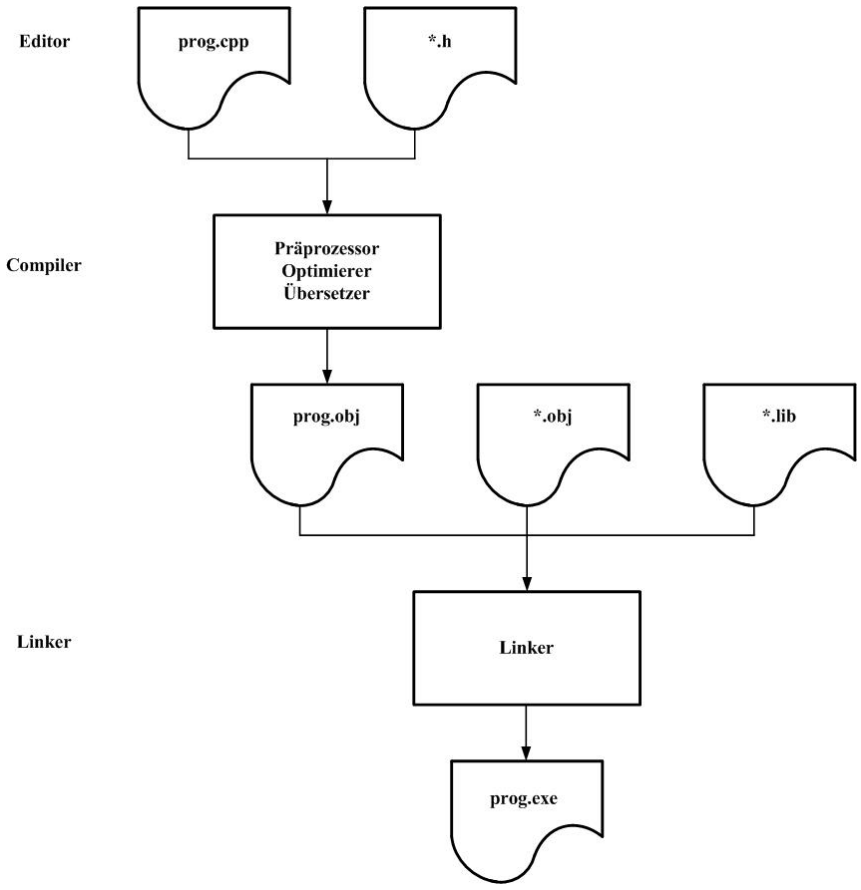
[7] ACSII engl. Abk. American Standard Code for Information Interchange

[8] nach gr. syntaktikós, den korrekten Satzbau

Anschließend wird der Quellcode unter verschiedensten Aspekten optimiert. Erst zum Abschluss des Kompiliervorgangs wird der Maschinencode erzeugt, der in so genannten Objektdateien mit der Dateinamenserweiterung „.obj“ gespeichert wird.

3.2.3 Linker

Nachdem alle Quelltextdateien eines Programms kompiliert worden sind und als Objektdateien vorliegen, können diese mittels dem Linker zu einem fertigen und lauffähigen Programm zusammengesetzt werden. Genau wie der Compiler ist auch der Linker ein eigenständiges Programm. Sein Name ist aus seiner Aufgabe abgeleitet, vorhandene Programmfragmente zu einem fertigen Programm zu verbinden (engl. to link).



Ablauf der Programmerstellung

3.2.4 Debugger

Es gibt Fehler im Quelltext, die der Compiler im Gegensatz zu syntaktischen Fehlern unmöglich erkennen kann: semantische Fehler[9], die oft auch logische Fehler genannt werden. Hier kompiliert das Programm zunächst anstandslos, verhält sich dann zur Laufzeit aber anders als erwartet – um nicht zu sagen falsch. Es muss nun festgestellt werden, an welcher Stelle im Quelltext unsauber gearbeitet worden ist und wie der Fehler behoben werden kann. Solche Fehler werden auch gerne als Bugs[10] bezeichnet. Um die Suche nach solchen Fehlern zu erleichtern, gibt es ein spezielles Programm: der Debugger. Dieser erlaubt es, den Quelltext Zeile für Zeile und Anweisung für Anweisung durchzugehen und quasi in Zeitlupe zu beobachten, was gerade vor sich geht. In der Entwicklungsumgebung Borland C++ kann das Debuggen über den Menüpunkt (Start / Einzelanweisung / F7) ausgeführt werden.

Aufgabe: Führe eine Recherche zum Thema Debugging durch. Gibt es eine Erklärung, warum gerade dieser Begriff für die Fehlersuche in Programmen verwendet wird?

3.3 DEV-C++

Vollständige Multiplatform-IDEs sind umfangreiche Programmpakete, die neben Editor, Compiler, Linker und Debugger auch graphische Benutzerschnittstellen mitbringen. Für die Entwicklung von komplexen Programmen mit graphischer Oberfläche sind solche Entwicklungsumgebungen praktisch unerlässlich. Für einen Einsteiger hingegen, der die Grundlagen der ihm neuen Programmiersprache C++ lernen möchte, sind die oft nur reiner Ballast.

[9] semantisch: den Inhalt eines sprachlichen Zeichens betreffend

[10] bug (dt. Wanze; Insekt; Käfer)

Jetzt aber genug mit der Theorie. Wir wollen Dev-C++ kennen lernen, indem wir mit dem Programm arbeiten. Doppelklicke dazu folgendes Symbol auf der graphischen Benutzeroberfläche des Betriebssystems:

Viel Spaß beim Arbeiten und beim Erlernen der Programmiersprache C++!

4 Übungen und Lösungen

4.1 Zum Kapitel „Problemanalyse„

Istanalyse: Untersucht und beschreibt das vorliegende System durch Betrachtung der einzelnen Systemkomponenten, die durch den Einsatz einer Rechenanlage beeinflusst werden, ihrer Funktionen und ihres Zusammenwirkens.

Sollkonzeptentwicklung: Umfasst einerseits die Anforderungen an die Software, andererseits Hinweise auf die Umstrukturierung des übrigen Systems.

Durchführbarkeitsstudie: Gibt eine Aussage darüber, ob die entwickelten Vorstellungen über das Softwareprodukt mit den vorhandenen Mitteln überhaupt realisiert werden können, also ob sie technisch durchführbar, ökonomisch vertretbar und von den Mitarbeitern im Rahmen des jeweiligen Unternehmens umsetzbar sind.Zugleich versucht man mögliche Vor- und Nachteile aufzuzeigen.

Projektplanung: Hier erstellt man Zeitpläne, plant die Verteilung des Personals und ermittelt die erforderlichen Hilfsmittel. Der Projektplan sollte in einem Pflichtenheft festgehalten werden.

4.2 Zum Kapitel „Debugger“

Das Wort Debugging geht zurück auf den schon für das 19. Jh. in Amerika dokumentierten Gebrauch des engl. Wortes bug (dt. Wanze; Insekt; Käfer) für Fehler in einem technischen System. Zur Verbreitung in der Informatik trug wahrscheinlich folgende Begebenheit bei: Am 09. 09. 1947 fiel der von einem Team um Howard H. Aiken entwickelte Rechner MARK II aus. Bei der Fehlersuche entdeckten die Forscher in einem Relais eine tote

Motte, die die Ursache des Fehlers war. Sie wurde zusammen mit einer Notiz „ordnungsgemäß“ im Logbuch des MARK II festgehalten.

Von diesem Zeitpunkt an wurde zunächst in einem Team und dann in immer größerem Kreis das Beheben von Hard- und Softwarefehlern als Debugging bezeichnet. Das Logbuch des MARK II mit der eingeklebten Motte kann noch heute im National Museum of American History in Washington, D.C.[11], besichtigt werden.

[11] D.C. steht für District of Columbia, der amerikanische Regierungsbezirk.

Literaturverzeichnis

Duden, Schülerduden Informatik, 4. Aktualisierte Auflage, ISBN 3-411-04484-5