

Sich einem Computer mitteilen

Benus Becker

Das Problem und der Computer

Das Ziel beim Programmieren ist es, eine systematische, schematisch ausführbare Verarbeitungsvorschrift / Handlungsanweisung für den Computer zu finden, die aus einer Eingabe eine von dieser abhängige Ausgabe berechnet, wobei die in der Ein- und Ausgabe enthaltenen Daten Informationen über die reale Welt enthalten.

Programmiersprachen sind ein Medium, einem Computer mitzuteilen, wie er das zu tun hat. Computer sind an sich sehr dumm, und die Sprache, die sie verstehen, ist sehr beschränkt. Es ist also nötig, die Formulierung der Problemstellung und -lösung an das Verständnis eines Computers anzupassen.

Programmieren heißt, die Lösung eines Problems beschreiben. Dazu muss man es erst einmal selbst so präzise verstanden haben, daß man es unmissverständlich formulieren kann, denn Computern fehlt der Befehl **tu-was-ich-meine-und-nicht-was-ich-sage**. Um ans Ziel zu gelangen, also vom Computer die Lösung eines Problems berechnen zu lassen, muss man ihm für jede möglicherweise auftretende Situation durch eine Menge unter Umständen alternativer Handlungsanweisungen vorschreiben, wie er dabei vorgehen soll.

Wichtig ist auch: ein Computer befindet sich immer genau in *einem* Zustand. Das heißt erstens, sein Speicher ist ständig in einem eindeutigen Zustand. Jeder Wert, den er kennt (zum Beispiel der Wert, der einer Variablen zugewiesen ist), ist immer eindeutig. Und zweitens befindet er sich beim Durchlaufen eines Programms immer genau an einer Stelle. Und von dieser entscheidet er anhand seines eindeutigen Zustands, wie er weiter zu handeln beliebt.

Divide et Impera

Das wohl wichtigste Paradigma des Programmierens, also die häufigste, eigentlich die einzige Herangehensweise beim Programmieren ist die Modularität. Es gibt viele Programmiersprachen, die teilweise sehr unterschiedlichen Stilen folgen (imperativ, funktional, objektorientiert, logisch, ...), deren Zweck es aber immer ist, wenn eben auch auf sehr unterschiedliche Weise, dem Benutzer, dem Programmierer also, die Möglichkeit zu geben, seine Problemstellung, also das, was er eigentlich sagen will, in kleinere Problemstellungen aufzuteilen, und deren Lösungen — welchen wiederum auf dem Weg der Unterteilung entstanden sein können — sinnvoll miteinander so zu kombinieren, dass sich die eigentliche angestrebte Lösung ergibt.

Dieser Art der Problemlösung entspricht auch eine kognitionswissenschaftliche Erkenntnis: niemals versucht man ein Problem im Ganzen zu lösen; um

das kanonische Beispiel aufzugreifen — der Vorgang des Busfahrens als Ganzes ist nicht lösbar; erst durch Hintereinanderausführen der Teilprobleme, die einfacher sind, ist Busfahren möglich: erst Jacke anziehen, Tür auf, rausgehen, Tür wieder zu; den Schlüssel nicht vergessen haben, zur Bushaltestelle laufen, warten, einsteigen, eine Karte kaufen und dann hinsetzen, wenn das die Umstände erlauben.

Das Ziel ist es, dass diese neu entstandenen, einfacheren Probleme elementar genug sind (trivial), um direkt ausgeführt werden zu können. Andernfalls müssen sie wieder weiter aufgeteilt werden.

Ein Computer denkt sich leider/gottseidank nichts. Das hat zur Folge, dass man ihm im Prinzip einen Algorithmus bis ins letzte Detail (und das sind die Operationen, die der Prozessor ausführen kann: Addition, Gleichheitstest und einen Sprung an eine andere Stelle im Programm) erklären muss. Deswegen ist die genannte Modularität unbedingt nötig, denn man muss beim Programmieren auf schon vorhandene Befehle und Funktionen zugreifen, wenn man nicht komplett in Maschinsprache schreiben will.

Die eine Aufgabe beim Programmieren ist es also, sich eine sinnvolle Aufteilung des eigentlichen, großen Problems in kleinere Probleme auszudenken, diese kleineren Probleme unabhängig voneinander zu lösen und deren Lösungen zum eigentlich gesuchten Gesamtergebnis wieder zusammenzuführen.

Solch ein von einem Algorithmus zu lösendes Problem basiert in den meisten Fällen auf einem Teil der Eigenschaften der realen Welt (nämlich genau den Eigenschaften, die für das Problem relevant sind). Einen Weg zu finden, diese Eigenschaften im Computer zu repräsentieren, der einem auch hier wieder nur sehr eingeschränkte Mittel zur Verfügung stellt, im Grunde nämlich 0en und 1en, ist die andere wichtige Aufgaben, der man sich beim Programmieren stellen muss.

Viele der in der Informatik verwendeten Strukturen, um Daten aus der realen Welt im Computer darzustellen, sind der Mathematik entliehen. So können für einen Algorithmus relevante, kontinuierliche Eigenschaften der realen Welt (zum Beispiel eine Temperatur oder ein Winkel) durch reelle Zahlen modelliert werden, wohingegen ganze Zahlen dafür weniger geeignet wären.

Die Tätigkeit des Programmierens stellt also zwei eng verbundene Anforderungen: sich eine für den Computer geeignete Repräsentation des Problems und dessen Lösung auszudenken und das Problem solange in Teilprobleme aufzuteilen, bis diese trivial sind oder schon anderweitig gelöst wurden.

Programmiersprachen und natürliche Sprachen

Sprache ist, aus der Endlichkeit ihrer Bestandteile (Wörter) und ihrer eigenen Unendlichkeit folgend, hierarchisch (also durch eine Grammatik) strukturiert. Im Deutschen wird der Zusammenhang von Strukturen durch Satzstellung und vor allem durch Übereinstimmungen von Worteeigenschaften (Fall, Geschlecht, etc.) entschieden:

Der Bauer gibt dem Esel etwas Heu.

Strukturierung eines Satzes anstatt auf gewohnter, morphologischer Ebene ist auch denkbar durch Klammerungen, die die engere Zusammengehörigkeit von Satzteilen kennzeichnen:

(geben Bauer Esel (etwas Heu))

Damit ist schon die einzige syntaktische Regel für die *Wohlgeformtheit* in Lisp gegeben: Wohlgeformt sind sogenannte Atome, natürlichsprachlich bekannt als „Wörter“, und Listen von wohlgeformten Lisp-Ausdrücken, also Folgen von Atomen oder weiteren Listen zwischen Klammern „(“ und „)“. Es werden also Lisp-Ausdrücke durch die vielen Klammern, die im Lisp-Code vorkommen, strukturiert, wie das in natürlichen Sprachen der Fall ist mit morphologischen Eigenschaften.

Allerdings ist nicht jeder wohlgeformte Lisp-Ausdruck, wie er gerade definiert wurde, ein *korrekter* Lisp-Ausdruck: Genau wie in natürlichen Sprachen regieren manche Lisp-Ausdrücke bestimmte Eigenschaften von syntaktisch untergeordneten Ausdrücken. Wie in dem Satz „Peter läuft den See.“ das Verb „laufen“ eigentlich kein Akkusativobjekt erlaubt, ist es für manche Lisp-Ausdrücke festgelegt, welche Eigenschaften ihre syntaktische Umgebung, also die ihnen untergeordneten Ausdrücke, haben muss. Warum also `(length 1 2 3)` keinen Sinn macht, `(length '(1 2 3))` dafür umso mehr, sollte im Lauf des Semesters klar werden.

Absichtserklärung

Das Ziel dieses Artikels ist es, ein Gefühl dafür zu geben, was jemand tut, der angibt zu programmieren. Es ist *nicht* das Ziel dieses Artikels, tiefe Einsichten oder anwendbares Wissen zu vermitteln — das passt nicht auf drei Seiten. Es sind nur ein paar Ansichten aufgeschrieben von denen zu wissen es vielleicht wert ist, wenn man zu programmieren anfängt. Obwohl manches vielleicht noch etwas unverständlich wirkt, lohnt es sich, wenigstens davon gehört zu haben, weil es vom Gehirn trotzdem weiterbearbeitet wird, und später, wenn das alles etwas mehr zusammengewachsen ist, Sinn bekommt.
