IN DIESEM KAPITEL

Wie ergänzen sich Mensch und Maschine?

Welche Schritte werden beim Erstellen eines Programms durchlaufen?

Kapitel 1
Von der Idee zum Programm

Dieses Kapitel skizziert den ganzen Weg, der bei der Erstellung und Benutzung eines Computerprogramms beschritten werden muss: von der Idee über den Algorithmus bis hin zur schlussendlichen Ausführung der fertigen Anwendung.

Mensch vs. Maschine

Computer wurden erschaffen, um Menschen beim Rechnen zu helfen. Heute geht das mit einem handlichen Taschenrechner oder einer App auf dem Smartphone, während früher einfache Rechner schnell einmal ganze Räume ausgefüllt haben.

Mit der Zeit wurden Rechner immer kleiner – und gleichzeitig leistungsstärker. Die heutigen modernen Computer unterstützen den Menschen nicht nur bei einfachen Berechnungen, sondern bei so ziemlich allen Problemen des menschlichen Alltags und des Berufslebens:

- ✔ Wie viel Steuern muss man bei einem Brutto-Einkommen von 25.000 Euro zahlen? (Steuerberater)
- ✔ Wie viele Kalorien habe ich bei meinem heutigen Training verbrannt? (Sportler)
- ✔ Wie sieht die aktuelle Tabelle der Fußball-Bundesliga aus? (Fußballfan)
- ✔ Wenn ich ein Haus mit den angegebenen Materialien in der angegebenen Form baue (oder bauen lasse), ist das stabil oder stürzt es ein? (Architekt)
- ✔ Wie komme ich am schnellsten mit dem Auto vom Münchner Hauptbahnhof zum Rathaus von Wladiwostok? (Taxifahrer)

All diese Fragen ließen sich auch ohne die Hilfe einer Maschine klären. Für jede der Fragen kann der Mensch sich die genauen Arbeitsschritte überlegen, die zur Beantwortung nötig wären. Diese müsste der Mensch dann selbst durchführen – was vermutlich sehr viel langsamer und potenziell auch fehleranfälliger ist, als wenn ein Computer diese Aufgaben durchführt.

Mensch und Maschine haben also sehr unterschiedliche Stärken und Schwächen (siehe Abbildung 1.1). Während Menschen sehr kreativ sind und neue Lösungswege für Problemstellungen finden können, sind sie auch langsam und fehleranfällig bei der Durchführung. Computer dagegen können gestellte Aufgaben sehr schnell und zuverlässig lösen, führen aber nur exakt die vorgegebenen Schritte aus. Computer »denken« nicht mit, sondern arbeiten streng nach Vorschrift.

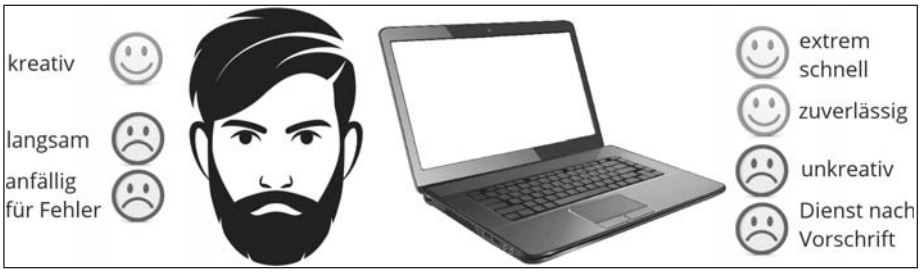

Abbildung 1.1: Mensch vs. Maschine

Um das beste Resultat zu erzielen, müssen Mensch und Maschine Hand in Hand arbeiten: Beide müssen ihre eigenen Stärken einbringen, um die Schwächen des jeweils anderen auszugleichen. Die optimale Strategie zur Lösung eines Problems ist demnach wie folgt:

1. Zunächst muss das zu lösende Problem vom Menschen genau spezifiziert werden: Welche Voraussetzungen sind gegeben und was genau soll herausgefunden werden? (*Problemstellung*)

2. Der Mensch findet heraus, welche Schritte zum Lösen des Problems vonnöten sind. (*Algorithmus*)

3. Der Mensch instruiert den Computer, sodass dieser später in der Lage ist, die einzelnen Schritte des Algorithmus eigenständig durchzuführen. (*Programmierung*)

4. Der Computer kann nun jederzeit dazu aufgefordert werden, die zuvor instruierten Schritte durchzuführen, und damit das ursprüngliche Problem lösen. (*Programmausführung*)

Bevor Sie einen Computer dazu verwenden können, ein Problem zu lösen, müssen Sie also erst einmal selbst wissen, wie sich das Problem generell lösen lässt. Das ist eine wichtige Voraussetzung für die Programmierung und in vielen Fällen nicht trivial. Das Finden eines geeigneten Algorithmus kann unter Umständen viel aufwendiger sein als die eigentliche Programmierung selbst.

Auch als Programmierer genügt es nicht, einfach zum Computer zu sagen: »Löse dieses Problem mal für mich.« Das wird den Rechner nicht sonderlich beeindrucken oder motivieren, sondern er wird die Aufforderung schlichtweg ignorieren.

Natürlich könnten Sie einen digitalen Assistenten wie Google Now, Siri, Cortana oder Alexa nutzen und diesen nach der Lösung Ihres Problems fragen (beispielsweise nach der Tabelle der Fußball-Bundesliga). Diese Assistenten sind

aber auch nur Programme, die so entworfen wurden, dass sie natürliche Sprache teilweise interpretieren und bestimmte, auf diese Weise formulierte Aufgaben lösen können. Hier hat Ihnen jemand die Schritte 1 bis 3 bereits abgenommen.

In diesem Fall sind Sie nur der Anwender des Programms. Stattdessen möchten Sie aber lernen, wie Sie selbst Programme erstellen können, die derartige Probleme zu lösen in der Lage sind. Oder nicht?

Einen Algorithmus entwickeln

Ein Algorithmus ist ganz allgemein eine eindeutige Handlungsvorschrift zum Lösen eines Problems. Da sehr unterschiedliche Arten von Problemen mit Computern gelöst werden, gibt es keine allgemeingültige Anleitung zum Erstellen eines Algorithmus.

Als Anfänger in der Programmierung werden Sie natürlich zunächst nur eher kleinere und einfachere Probleme lösen, die keine komplizierten Algorithmen benötigen. Dennoch macht es Sinn, sich jetzt schon einmal kurz mit der Erstellung von Algorithmen zu befassen.

Eine Problemstellung und der zugehörige Algorithmus zur Lösung sollten immer möglichst allgemein formuliert werden. Ein Algorithmus, der zwei beliebige Zahlen addieren kann, ist besser als ein Algorithmus, der immer nur 08 + 15 rechnet.

Ein Algorithmus wird daher in der Regel einen oder mehrere Eingabe-*Parameter* erwarten und eines oder mehrere Ergebnisse liefern. Das Ergebnis ist die sogenannte Ausgabe. Der Algorithmus zur Addition von zwei Zahlen würde zum Beispiel die beiden zu addierenden Zahlen als Parameter erhalten und als Ergebnis die Summe liefern. Er kann so mehrmals mit unterschiedlichen Parametern ausgeführt werden, um unterschiedliche Summen zu berechnen.

Im Übrigen brauchen Sie nicht wirklich einen Algorithmus zur Addition von zwei Zahlen zu erstellen. Beim Erstellen eines Algorithmus dürfen Sie davon ausgehen, dass der Computer, der den Algorithmus später ausführen soll, einige gewisse »Grundkenntnisse« bereits mitbringt. Dazu zählen die vier Grundrechenarten: Das Summieren, Subtrahieren, Multiplizieren und Dividieren beherrscht jeder Computer aus dem Effeff.

Damit Sie eine erste Idee davon bekommen, wie ein Algorithmus aussieht, zeige ich Ihnen hier ein kurzes Beispiel.

Problem: Wie ist die Gesamtsumme einer Reihe von Zahlen, die sich in einer Liste befinden?

Als Beispiel verwende ich hier ein relativ alltägliches Problem. Sie erhalten eine Liste von Zahlen und sollen deren Gesamtsumme bestimmen.

Diesem Problem werden Sie in Ihrem Alltag regelmäßig gegenüberstehen, zum Beispiel wenn Sie beim Kaufen an einer Kasse von der Kassiererin Wechselgeld erhalten oder wenn Sie beim Backen das Gesamtgewicht aller verwendeten Zutaten bestimmen wollen.

Der zu erstellende Algorithmus erhält als Eingabe-Parameter also eine Liste von Zahlen, wobei die Anzahl der enthaltenen Elemente variabel ist. Das bedeutet, jedes Mal, wenn Sie

das Problem lösen sollen, handelt es sich um eine unterschiedliche Liste mit einer unterschiedlichen Anzahl an Elementen.

Ich nenne diese Liste im Folgenden `L`. Sie soll `n` Elemente besitzen, die mit z_1 bis z_n bezeichnet werden: $L = [z_1, z_2, \ldots, z_n]$

Wie würden Sie ein solches Problem manuell lösen, wenn Sie keinen Computer haben, der Ihnen dabei hilft?

- ✔ Eine Idee wäre, sich eine Zwischensumme zu merken und zu dieser nacheinander alle Elemente der Liste hinzuzuaddieren.
- ✔ Zunächst wäre die Zwischensumme 0. Dann würde das erste Element der Liste dazu addiert, danach das zweite und so weiter.
- ✔ Wenn auf diese Weise alle Elemente der Liste durchlaufen wurden, enthält die Zwischensumme die Gesamtsumme aller enthaltenen Elemente, also das gesuchte Endergebnis.

Im Groben ist dies bereits der Algorithmus, mit dem man die Summe von beliebigen (Zahlen-)Listen berechnen kann. Dieser muss jetzt nur noch etwas ausführlicher aufgeschrieben werden.

Zunächst fällt noch auf, dass für die Lösung des Problems eine Zwischensumme verwendet wurde. Innerhalb eines Algorithmus würde man dafür eine sogenannte *Variable* verwenden.

Eine Variable ist ein Behälter für eine veränderliche Größe, die während der Ausführung eines Algorithmus verwendet wird. In vielen Fällen werden Zahlen in Variablen abgelegt. In Variablen können aber auch Texte stehen. Sie können eine Variable mit einem Wert belegen und diesen Wert jederzeit wieder auslesen oder abändern. Innerhalb eines Algorithmus werden Variablen zum Beispiel verwendet, um Zwischenergebnisse abzuspeichern.

Sie können Variablen beliebige Namen geben, zum Beispiel `arxqt22` oder `hrmpfl44mop`. Sehr viel sinnvoller ist es aber, die Variablen so zu benennen, dass man erkennen kann, wozu man sie benutzt. Über den Namen einer Variablen kann bei Bedarf auf den Variablenwert zugegriffen werden.

Mit diesem Wissen im Hinterkopf lässt sich nun ein Algorithmus zum Berechnen der Gesamtsumme einer Liste von Zahlen entwickeln. Der Algorithmus müsste die folgenden Arbeitsschritte enthalten:

1. Die Liste $L = [z_1, z_2, \ldots, z_n]$ wird an den Algorithmus übergeben.
2. Es wird eine neue Variable mit dem Namen `summe` angelegt. In dieser wird der Wert 0 gespeichert.
3. Solange die Liste `L` noch Elemente enthält (also nicht leer ist), führe die folgenden Schritte aus:
   - Lege eine Variable `element` an und speichere den Wert des ersten Elements der Liste `L` darin.

- Entferne das erste Element aus der Liste `L`.
- Bilde die Summe der Werte, die in den Variablen `summe` und `element` gespeichert sind, und speichere das Ergebnis in der Variablen `summe`.

  Der Wert, der zuvor in der Variablen `summe` gespeichert war, wird also mit einem neuen Wert überschrieben.

4. Beende die Ausführung des Algorithmus und liefere den Wert, der in der Variablen `summe` gespeichert ist, als Endergebnis.

Der Algorithmus macht genau das, was zuvor beschrieben wurde. Es wird eine Variable angelegt, in der das Zwischenergebnis der bisherigen Summation abgespeichert wird. Zu Beginn ist dieses 0.

Entscheidend ist der Schritt 3 des Algorithmus. Dieser wird mehrmals ausgeführt, nämlich solange die Liste noch Elemente enthält.

- ✔ Bei jedem Durchlauf von Schritt 3 wird der Wert des ersten Elements der Liste bestimmt und in einer eigenen Variablen gesichert.
- ✔ Es wird nun die Summe aus dem bisherigen Zwischenergebnis und dem soeben gesicherten Wert berechnet.
- ✔ Zudem wird das erste Element aus der Liste entfernt, sie schrumpft also.

Der Algorithmus verarbeitet demnach jeweils das erste Element der Liste und entfernt es dann daraus. Wenn die Liste irgendwann leer ist, wurden folglich alle Elemente bereits verarbeitet, das heißt, ihr Wert ist in der Gesamtsumme enthalten. Der Algorithmus wird dann beendet. Man sagt auch, er *terminiert*.

Ich werde den Ablauf des Algorithmus abschließend noch an einem Beispiel demonstrieren. Mithilfe des vorgestellten Algorithmus soll die Summe der Elemente in der Liste `L = [12, 34, 18]` bestimmt werden.

1. Der Algorithmus erhält die Liste `L = [12, 34, 18]` als Eingabeparameter.
2. Es wird eine Variable `summe` angelegt. Diese erhält den Wert 0.

   `summe` = 0
3. Es wird geprüft, ob die Liste `L` leer ist. Da das nicht der Fall ist (sie enthält drei Elemente), werden die drei Unterschritte ausgeführt.

   - Schritt 3.1: Eine Variable `element` wird angelegt, und der Wert 12 (der Wert des ersten Elements der Liste) wird darin gespeichert.

     `element` = 12
   - Schritt 3.2: Das erste Element der Liste L wird entfernt. Sie hat nur noch zwei Elemente.

     `L = [34, 18]`

- Schritt 3.3: Die Summe von `summe` (0) und `element` (12) wird berechnet und in der Variablen `summe` gespeichert.

  ```
  summe = 12
  ```

  Schritt 3: Die Liste `L` enthält noch zwei Elemente, ist also immer noch nicht leer. Daher werden die drei Unterschritte erneut ausgeführt.

- Schritt 3.1: In der Variablen `element` wird der Wert 34 gespeichert.

  ```
  element = 34
  ```

- Schritt 3.2: Das erste Element der Liste `L` wird entfernt. Sie hat nun nur noch ein Element.

  ```
  L = [18]
  ```

- Schritt 3.3: Die Summe von `summe` (12) und `element` (34) wird berechnet und in der Variablen `summe` gespeichert.

  ```
  summe = 46
  ```

  Schritt 3: Die Liste `L` enthält immer noch ein Element. Da sie nicht leer ist, werden die drei Unterschritte noch einmal ausgeführt.

- Schritt 3.1: In der Variablen `element` wird der Wert 18 gespeichert.

  ```
  element = 18
  ```

- Schritt 3.2: Das erste Element der Liste `L` wird entfernt. Sie ist nun leer.

  ```
  L = []
  ```

- Schritt 3.3: Die Summe von `summe` (46) und `element` (18) wird berechnet und in der Variablen `summe` gespeichert.

  ```
  summe = 64
  ```

  Schritt 3: Die Liste L ist jetzt leer. Daher werden die Unterschritte nicht mehr ausgeführt.

4. Der Wert der Variablen `summe`, 64, wird als Endergebnis des Algorithmus geliefert.

Das ist das richtige Ergebnis, denn 12 + 34 + 18 ergibt 64.

In diesem Abschnitt haben Sie jetzt ein erstes Beispiel für einen Algorithmus gesehen. In Kapitel 5 werde ich Ihnen einige weitere, etwas komplexere Algorithmen zeigen.

Wenn Sie einen Algorithmus zur Lösung eines Problems erstellen sollen, sollten Sie ähnlich wie im gezeigten Beispiel vorgehen.

Überlegen Sie sich zunächst, wie Sie ein einfaches Beispiel des Problems von Hand lösen würden. Machen Sie sich jeden einzelnen Arbeitsschritt klar. Versuchen Sie danach, die ausgeführten Schritte zu verallgemeinern und so den Algorithmus zu entwickeln.

Mit dem Rechner kommunizieren

Haben Sie erst einmal einen funktionierenden Algorithmus erstellt, so haben Sie bereits einen großen Schritt in Richtung eines lauffähigen Programms getan. Dennoch sind Sie noch nicht am Ziel. Den in einer natürlichen Sprache (wie Deutsch oder Englisch) verfassten Algorithmus kann der Computer nicht verstehen – und damit auch nicht ausführen.

Toll wäre es ja, wenn der Computer das könnte: gesprochene (oder geschriebene) Anweisungen verstehen und in die Tat umsetzen. Dann könnte jeder, der der deutschen (oder einer anderen) Sprache mächtig ist, eigene Programme erstellen, indem er dem Computer die auszuführenden Schritte diktiert.

Noch besser wäre es, wenn der Computer »mitdenken« könnte und allein aus der Beschreibung eines Problems die korrekte Lösung ableiten und ausführen würde. Allerdings gibt es auch angesehene Wissenschaftler (wie beispielsweise Stephen Hawking), die vor genau diesem Szenario warnen: einem Computer mit echter Intelligenz, der sich letztendlich gegen seine Erschaffer wenden könnte.

Vorerst ist das aber noch kein Thema – weder muss in nächster Zeit mit einem Angriff von Skynet gerechnet werden noch mit einem Arnold Schwarzenegger, der die Welt in Schutt und Asche legt. Das bedeutet aber auch, dass es nicht genügt, einen Algorithmus in natürlicher Sprache zu formulieren. Computer verstehen nur ihre eigene, ganz spezielle Sprache – in diese muss der Algorithmus übersetzt werden.

Maschinencode

Diese Sprache, in der man mit dem Computer kommunizieren kann, nennt sich *Maschinencode*. Der Maschinencode kennt nur sehr einfache Befehle, wie zum Beispiel das Addieren von zwei Zahlen oder das Prüfen, ob eine Zahl größer als eine andere ist. Es handelt sich also um eine sehr primitive Sprache, vor allem im Vergleich zu natürlichen Sprachen wie Deutsch oder Englisch.

Einen Algorithmus direkt von einer natürlichen Sprache in Maschinencode zu übersetzen, wäre eine sehr mühselige Arbeit. Jede aufwendigere Operation des Algorithmus müsste aus den sehr primitiven Möglichkeiten des Maschinencodes zusammengebastelt werden. Das wäre in etwa so, als wenn Sie nur mithilfe von Nägeln, Styropor und einem Hammer einen Airbus A380 zusammenbauen müssten. Computer wären nicht so populär, wenn es keine bessere Möglichkeit der Programmierung gäbe, als Maschinencode zu schreiben.

Programmiersprachen

Als Zwischenschritt zwischen Maschinencode und natürlicher Sprache wurden deshalb die Programmiersprachen eingeführt. Programmiersprachen sind zwar auch deutlich einfacher als natürliche Sprachen (vor allem in Bezug auf Grammatik und Wortschatz), aber grundsätzlich noch verständlich. Um es mit der Metapher zu sagen: Der Airbus A380 setzt sich jetzt zwar immer noch nicht durch alleiniges Anschreien der Materialien wie von Geisterhand zusammen, aber immerhin hätten Sie jetzt Maschinen, eine Fabrik und tausend Arbeiter, die Ihnen helfen.

Programmiersprachen können sowohl vom Computer (mit gewisser Hilfe) als auch vom Menschen verstanden werden – und sind damit die optimale Wahl, um eine Kommunikation zwischen Mensch und Maschine zu ermöglichen.

Die Aufgabe eines Programmierers ist es, die verbale Beschreibung eines Algorithmus in eine Programmiersprache zu transformieren. Den dabei entstehenden Text nennt man *Quellcode*. Der Quellcode entspricht also der Übersetzung des Algorithmus in die gewählte Programmiersprache. So wie Sie deutschen Text in englischen Text übersetzen können, können Sie einen Algorithmus in einen Quellcode übersetzen. Na ja, in etwa …

Wie Quellcode aussieht, werden Sie in den folgenden Kapiteln sehen. Der Quellcode ist jedoch wie zuvor beschrieben nur ein Zwischenschritt – denn der Computer versteht ohne zusätzliche Hilfe weiterhin nur den Maschinencode.

Zu jeder Programmiersprache gibt es deshalb einen sogenannten *Compiler* (zu Deutsch: Übersetzer). Der Compiler ist selbst ein Programm, das in der Lage ist, den Code, der in einer bestimmten Programmiersprache geschrieben wurde, in Maschinencode zu übersetzen. Üblicherweise wird der Compiler vom Entwickler einer Programmiersprache zur Verfügung gestellt.

Sobald Sie den Quellcode eines Programms fertiggestellt haben, können Sie diesen mithilfe des Compilers in Maschinencode umwandeln.

Damit sind alle Schritte komplett. Ein Problem kann wie folgt gelöst werden:

1. Das zu lösende Problem muss von einem Menschen genau bestimmt werden.

   Welche Eingaben sind vorhanden und was soll daraus bestimmt werden?

2. Nun entwickelt der Mensch eine allgemeine Strategie, wie das Problem gelöst werden kann.

   Diese Strategie wird in einzelne, kleine Schritte unterteilt, die einen Algorithmus zur Lösung des Problems beschreiben.

3. Der verbal formulierte Algorithmus muss jetzt in eine strukturiertere Form gebracht werden.

   Dafür wird eine Programmiersprache verwendet. Mit einer Programmiersprache können die Schritte, die zur Lösung eines Problems vonnöten sind, mit dem sogenannten Quellcode ausgedrückt werden.

4. Es wird ein Compiler benutzt, der den Quellcode in den Maschinencode übersetzt.

   Der Compiler ist ein Programm, das vom Ersteller der Programmiersprache zur Verfügung gestellt wird. Der Programmierer kann seinen Programmcode also dem Compiler als Eingabe übergeben und erhält als Ergebnis ein fertiges, ausführbares Programm, das aus Maschinencode besteht.

5. Das fertige Programm, das als Maschinencode vorliegt, kann nun jederzeit und wiederholt von einem Rechner ausgeführt werden.

Das Wichtigste in Kürze

- ✔ Menschen sind im Vergleich zu Computern zwar kreativ, aber auch langsam und fehleranfällig.
- ✔ Computer dagegen arbeiten schnell und sicher, allerdings nur exakt nach Anweisung.
- ✔ Mensch und Maschine arbeiten optimal zusammen, wenn Menschen die auszuführenden Schritte zur Lösung eines Problems herausarbeiten und vom Computer durchführen lassen.
- ✔ Ein Algorithmus ist eine eindeutige Handlungsvorschrift zum Lösen eines Problems.
- ✔ Algorithmen sollten möglichst allgemein gehalten werden.
- ✔ Computer können nur den sehr schwer verständlichen Maschinencode verarbeiten.
- ✔ Programmiersprachen wurden als Zwischenschritt zwischen natürlicher Sprache und Maschinencode eingeführt, um Menschen die Programmierung von Rechnern zu erleichtern.
- ✔ Der in einer Programmiersprache geschriebene Quellcode muss mithilfe eines Compilers in Maschinencode übersetzt werden, bevor er von einem Rechner ausgeführt werden kann.

Übungen

Alle Übungen inklusive der Lösungsvorschläge finden Sie auch auf der Webseite zum Buch unter `https://www.wiley-vch.de/ISBN9783527718511`.

1. Zählen Sie weitere Probleme auf, die typischerweise von einem Computer gelöst werden.

2. Der Algorithmus zum Berechnen der Gesamtsumme aller Elemente einer Liste soll nun auf die Liste `L = [1, 3, 2, 4, 6]` angewendet werden.

Gehen Sie alle Schritte, die der Algorithmus durchführt, einzeln durch.

3. Erstellen Sie einen Algorithmus, der das Produkt aller Zahlen, die sich in einer Liste `L` befinden, berechnet.

Für die Liste `L = [2, 6, 10]` berechnet sich das Produkt zum Beispiel als $2 \cdot 6 \cdot 10 = 120$.

4. Nehmen Sie an, dass der Computer Zahlen nur addieren kann. Wie können dennoch ganzzahlige Multiplikationen durchgeführt werden?