Programmier-Empfehlungen

Bundeswettbewerb Informatik
H. Klaeren*

21. September 1995

> It is easier to write an incorrect program than understand a correct one.
>
> A. Perlis, Epigrams on Programming, No. 7

> Perhaps if we wrote programs from childhood on, as adults we'd be able to read them.
>
> A. Perlis, Epigrams on Programming, No. 24

Wie jede Sprache hat auch eine Programmiersprache drei Aspekte: *Syntax*, *Semantik* und *Pragmatik*. Unter diesen Aspekten werden folgende Fragen beantwortet:

Syntax Beim Schreiben: Welches sind die Regeln des Satzbaus, nach denen ich meinen Text aufbauen muß? Beim Lesen: Ist der gegebene Text grammatikalisch richtig? Welches sind seine Bestandteile im Sinne des korrekten Satzbaus?

Semantik Beim Schreiben: Welchen Bedeutungsinhalt will ich vermitteln? Beim Lesen: Was will der gegebene Text aussagen?

Pragmatik Beim Schreiben: Wie drücke ich das, was ich aussagen will, am besten aus? Was weiß ich über den Leser, in welchem Ton will ich ihn ansprechen? Beim Lesen: Wie sagt der Text das, was er sagen will? Welche zusätzliche Information über die Absicht des Schreibers kann ich aus der Art der Formulierung ableiten?

Im Bereich der Pragmatik sind Fragen des *Stils* angesiedelt; im Deutschunterricht wird diesen Fragen großes Gewicht beigemessen. Offensichtlich sind Syntax und Semantik eher die niederen Ebenen einer sprachlichen Kommunikation, während Pragmatik auf einer höheren Ebene diskutiert wird. Viele Programmierer glauben jedoch, daß man sich bei Programmiersprachen auf Syntax und Semantik allein beschränken kann, während Fragen des Stils praktisch überhaupt keine Rolle spielen. Dies mag dadurch begründet sein, daß der Computer sich darauf beschränkt, die Syntax der eingegebenen Programme zu überprüfen und die entsprechenden semantischen Aktionen auszuführen, während er pragmatische Aspekte gar nicht beurteilen kann.

In Wirklichkeit ist diese Einstellung jedoch falsch, denn:

> *Wir schreiben Programme nicht nur, damit ein Computer sie ausführt, sondern auch dafür, daß wir selbst oder andere Menschen sie verstehen können. Computer dagegen können Programme überhaupt nicht verstehen und brauchen dies auch nicht.*

Wir haben also Programme in der Hauptsache so zu schreiben, daß Menschen sie leicht lesen und verstehen können. Dann spielen aber Fragen der Pragmatik

*Eine Vorversion dieser Empfehlungen erschien unter dem Titel *Überlegungen zum Programmierstil* in *Computer und Unterricht*, Nr. 16, 1994, p. 52-54

automatisch die entscheidende Rolle! Der amerikanische Informatik-Pionier D. E. Knuth propagiert seit Jahren das sogenannte *„Literate Programming"*, also belesenes bzw. gebildetes Programmieren und stellt die provozierende Frage: „Wann haben Sie zuletzt ein gutes Programm gelesen?"

Verschiedene Programmiersprachen bedingen unterschiedliche Programmierstile; auch wollen wir nicht verhindern, daß man sich seinen persönlichen Stil heranbildet. Im folgenden geben wir deshalb nur minimale Empfehlungen, die auf alle Programmiersprachen zutreffen, auch wenn wir gelegentlich Pascal zu Beispielzwecken heranziehen. Wir formulieren diese Empfehlungen als *Regeln* und erinnern gleichzeitig daran, daß es keine Regel ohne Ausnahme gibt. Wo es nötig erscheint, geben wir eine kurze Begründung für die jeweilige Regel und manchmal auch einige Beispiele. Es kommt uns hauptsächlich darauf an, das Augenmerk auf die Bedeutung des Programmierstils zu lenken. Wichtig ist dann weniger, welcher Stil befolgt wird, als vielmehr, daß überhaupt erkennbar einem Stil gefolgt wird.

Eine sehr heikle Frage ist es, in welcher Sprache die Bezeichner und Kommentare des Programms aufgeschrieben werden sollen (Regel 1.4). Zunächst einmal scheint es gute Gründe zu geben, hierfür rigoros die englische Sprache zu verwenden: Die Schlüsselwörter der Programmiersprache, die sogenannten vordefinierten Bezeichner und die Bestandteile eventueller Programmbibliotheken sind sowieso englisch; deutsche Bezeichner und Kommentare in einem Programm führen also unweigerlich zu einem englisch-deutschen Kauderwelsch. Außerdem werden im Zeichen einer durch internationale Netzwerke immer näher zusammenrückenden Welt gute Programme schneller über die deutschen Grenzen hinweg ausgetauscht, als man glaubt. Andererseits sollte man den sowieso in der Computerwelt vorherrschenden Anglizismen nicht widerstandslos nachgeben und keine zusätzlichen intellektuellen Hürden für die Schüler aufbauen. Diese Frage muß jeder für sich selbst entscheiden, aber wir können sicher verlangen, daß er seine Entscheidung konsequent durchzieht, d.h. also entweder nur deutsche Bezeichner und Kommentare verwendet oder nur englische.

1 Bezeichner

> "When I use a word", Humpty Dumpty said in rather a scornful tone, "it means just what I choose it to mean — neither more nor less."
> "The question is," said Alice, "whether you *can* make words mean so many different things."
> "The question is," said Humpty Dumpty, "which is to be master — that's all."
>
> L. Caroll, Through the Looking Glass

> What's in a name? that which we call a rose,
> by any other name would smell as sweet.
>
> W. Shakespeare, Romeo and Juliet

> Schon gut! Nur muß man sich nicht allzu ängstlich quälen;
> Denn eben wo Begriffe fehlen,
> Da stellt ein Wort zur rechten Zeit sich ein.
>
> Goethe, Faust

Bezeichner oder *Namen* brauchen wir, um auf die in unseren Programmen eingeführten Objekte (Konstanten, Variablen, Prozeduren) oder allgemeinere Konzepte (Typen) zugreifen zu können. Die vorangehenden beiden Zitate legen dar, daß Namen immer eine heikle Angelegenheit sind; ein jeder Name ist nur ein kümmerlicher Ersatz für den dahinterstehenden *Begriff*. Umso wichtiger ist es, daß man nach passenden Namen sucht; man sollte für die Wahl seiner Bezeichner im Programm ähnliche Mühe und Phantasie verwenden, wie man sie bei der Wahl von Namen für Straßen, Plätze oder Haustiere aufbringt. Andererseits sollte man, nachdem man

einen Namen gewählt hat, auch konsequent dabei bleiben.

Regel 1.1 *Wähle kurze, klare, verständliche und treffende Namen für die Objekte deines Programms. Einbuchstabige Bezeichner wie* `i` *sind fast nur als Schleifenindizes oder formale Parameter von kurzen Prozeduren akzeptabel.*

Regel 1.2 *Wenn du einen Konstantenbezeichner als Namen für eine bestimmte Größe deklariert hast, verwende diese Größe nirgendwo anders mehr in expliziter Form, sondern nur noch unter der vereinbarten Bezeichnung. Strebe allgemein danach, möglichst alle konstanten Größen des Programms durch Konstantenbezeichner zu deklarieren.*

Begründung Wirkliche Konstanten wie etwa in der Mathematik oder Physik gibt es in der Informatik gar nicht; hier gilt vielmehr die scherzhafte Aussage, daß Konstanten etwas weniger variabel sind als Variablen. Wer eine Konstante deklariert, tut dies mit der Absicht, sie später zu ändern. Andererseits entsteht für fast alle Konstanten, die explizit in einem Programm auftauchen, der Wunsch nach ihrer Änderung oft schneller, als man denkt. Konstantenbezeichner bieten die Möglichkeit, solche Änderungen schnell und konsistent durchzuführen.

Regel 1.3 *Schreibe einen Bezeichner immer mit der gleichen Verteilung von Groß- und Kleinbuchstaben, auch wenn die Programmiersprache dies nicht verlangt.*

Begründung Moderne Programmiersprachen unterscheiden Groß- und Kleinbuchstaben sowieso, aber auch bei älteren Programmiersprachen kann der menschliche Leser verwirrt werden, wenn ein Bezeichner in unterschiedlichen Schreibweisen verwendet wird.

Regel 1.4 *Wähle in einem Programm entweder nur englische Wörter als Namen oder nur deutsche, aber nicht beides.*

Begründung Programme sollen auf den Leser einen logisch konsistenten Eindruck machen; dieser wird sicher getrübt, wenn der Programmierer sich nicht einmal konsequent in einer einzigen Sprache ausdrücken kann. Außerdem erschwert das ständige geistige Umschalten das Verständnis.

Regel 1.5 *Unterscheide deine Bezeichner typographisch von den Schlüsselwörtern der Programmiersprache. Wenn es die Programmiersprache erlaubt, schreibe Schlüsselwörter in Großbuchstaben und Bezeichner in Kleinbuchstaben.*

Begründung Untersuchungen zeigen, daß sich Text in Kleinbuchstaben leichter lesen läßt als in Großbuchstaben. Darum sollten prinzipiell also möglichst Kleinbuchstaben verwendet werden. Schlüsselwörter erfüllen jedoch eine ähnliche Funktion wie Wegweiser im Straßenverkehr; sie machen beim Lesen die Struktur des Programms deutlich und helfen dabei, sich zurechtzufinden. Deshalb sollen sie auffallen. Bei Veröffentlichungen von Programmen geschieht dies in der Regel, indem Schlüsselwörter fett gedruckt sind.

Regel 1.6 *Verwende das Unterstreichungszeichen „_“ nur im äußersten Notfall innerhalb von Bezeichnern.*

Begründung Typographisch unterscheidet sich das Unterstreichungszeichen nur wenig von einer Leerstelle; dieser Effekt kann durch Fotokopieren oder ein schlechtes Farbband noch unterstützt werden. Beim Lesen gerät man daher in eine Sackgasse, wenn man zunächst zwei Bezeichner zu sehen glaubt. Der gewünschte Effekt, die einzelnen Wörter eines zusammengesetzten Bezeichners voneinander zu trennen, kann durch einzelne Großbuchstaben erzielt werden.

Beispiel `receiveFiles`, `startServer`, `sendBuffer`

Regel 1.7 *Verwende*

Adjektive oder Partizipien	*für*	*boolesche Konstanten, Variablen und Funktionsprozeduren*
Nomina (Substantive)	*für*	*andere Konstanten, Datentypen, Variablen und Funktionsprozeduren*
Verben (im Imperativ)	*für*	*Prozeduren*

Begründung Boolesche Konstanten und Variablen bezeichnen Sachverhalte und Eigenschaften. Andere Konstanten und Variablen bezeichnen Dinge und Werte im allgemeinen; die entsprechenden Nomina können durch Attribute ergänzt werden. Prozeduren bezeichnen Tätigkeiten; wo notwendig, können Verben durch Objekte ergänzt werden. Funktionsprozeduren werden entsprechend den von ihnen abgelieferten Resultaten benannt.

Beispiele
```
removeTask(printQueue)
IF isTextFile(file) THEN readText(file)
WHILE tooHigh(speed) DO reduce(speed)
REPEAT ... UNTIL done
i := index(item,list)
```

2 Kommentare

> There, there, never mind, don't explain, I hate explanations; they fog a thing up so that you can't tell anything about it.
> M. Twain, A Connecticut Yankee at King Arthur's Court

Was keiner weiteren Erklärung bedarf, soll man auch nicht erklären. Programmierer sollen keine Zeilenschinder sein. Was jedoch für das Verständnis des Programms wichtig ist, soll man in Kommentaren erklären. Dann sollen es allerdings gute Erklärungen sein; auch Verweise auf die Literatur können hilfreich sein. Längere Abhandlungen gehören in eine separate Dokumentation.

Regel 2.1 *Füge hinter jedem (Funktions-) Prozedurkopf einen Kommentar an, der die Leistung dieser (Funktions-) Prozedur beschreibt. Hat die (Funktions-) Prozedur Parameter, so müssen diese in dem beschreibenden Text vorkommen. Bringe einen ähnlichen Kommentar für das Programm selbst an.*

Die nächsten beiden Regeln beziehen sich nur auf Pascal-ähnliche Sprachen:

Regel 2.2 *Steht das* `BEGIN` *einer (Funktions-) Prozedur weiter entfernt von der entsprechenden Kopfzeile, so füge hinter* `BEGIN` *einen Kommentar an, der klarmacht, was hier beginnt. Auch für das* `BEGIN` *des Hauptprogramms ist ein solcher Kommentar in der Regel nötig.*

Regel 2.3 *Ein* `END` *als schließende syntaktische Klammer erhält einen Kommentar, sobald es mehr als etwa fünf Zeilen von der entsprechenden öffnenden syntaktischen Klammer entfernt ist.*

3 Formatierung

> Heil'ge Ordnung, segenreiche
> Himmelstochter, die das Gleiche
> Frei und leicht und freudig bindet.
> F. von Schiller, Die Glocke

Programme sind meist hochkomplexe Texte, die nicht leicht zu verstehen sind. Im Gegensatz zu Texten der natürlichen Sprache sind sie aber stark strukturiert,

wobei die Strukturen z.T. beachtliche Tiefe erreichen. Wir können das Verständnis um Größenordnungen erleichtern, wenn wir die Struktur der Programmtexte unter Ausnutzung der zweidimensionalen Natur von Texten andeuten. Der von jedem Deutschlehrer sicher als unmöglich kritisierte Schachtelsatz:

```
      Der, der den, der den Pfahl, der in der Brücke, die nach Mainz
   führt, steckt, herausgezogen hat, anzeigt, erhält eine Belohnung.
```

wird z.B. leichter verständlich, wenn wir ihn so anordnen:

```
Der,
   der den,
      der den Pfahl,
         der in der Brücke,
            die nach Mainz führt,
         steckt,
      herausgezogen hat,
   anzeigt,
erhält eine Belohnung.
```

Regel 3.1 *Verwende Leerzeilen, um dein Programm in Abschnitte zu gliedern. Trenne besonders die einzelnen Prozeduren durch eine oder mehrere Leerzeilen. Auch Deklarationen komplizierter Datenstrukturen sollten durch Leerzeilen von der Umgebung abgesetzt werden.*

Regel 3.2 *Rücke strukturell untergeordneten Text gegenüber den höheren Ebenen ein.*
Strukturell auf gleicher Stufe stehenden Text stelle auf gleiche Einrückungsstufe. Wir empfehlen eine Einrückung um 2 bis 3 Stellen.

Beispiel

```
FUNCTION length(v: list): Integer;
(* Compute length of list v *)
BEGIN
   IF v = NIL THEN
      length := 0
   ELSE
      length := length(v↑.rest) +1
END (* length *)
```

Regel 3.3 *Füge vor und nach Operations- und Relationszeichen und dem Zuweisungszeichen je (mindestens) eine Leerstelle ein, aber nicht zwischen einem (Funktions-) Prozedurbezeichner und seiner Argumentliste und nicht zwischen einem Feldbezeichner und seinem Indexausdruck.*

4 Verschiedenes

> Le bon sens est la chose du monde la mieux partagée: car chacun pense en être si bien pourvu, que ceux même qui sont les plus difficiles à contenter en toute autre chose, n'ont point coutume d'en désirer plus qu'ils en ont.
>
> R. Descartes, Discours de la Méthode

> Get into a rut early: Do the same processes the same way. Accumulate idioms. Standardize. The only difference(!) between Shakespeare and you was the size of his idiom list — not the size of his vocabulary.
>
> A. Perlis, Epigrams on Programming, No. 10

Regel 4.1 *Gestalte dein Programm so lesbar und verständlich wie irgend möglich.*

Regel 4.2 *Verwende möglichst wenig globale Variablen; verwende sie vor allem nicht als Ersatz für Prozedurparameter.*

Begründung Wenn eine Prozedur globale Variablen liest und schreibt, so stellen diese implizite Parameter dar. Sofern die globale Variable nur gelesen wird, ist dies noch tolerierbar, obgleich jetzt die Parameter nicht mehr alle durch Inspektion des Prozedurkopfs erkennbar sind. Das Verändern globaler Variabler ist eine schwer durchschaubare Technik, die nur in Ausnahmefällen verwendet werden sollte und dann im Kommentar zum Prozedurkopf erwähnt werden sollte.

Regel 4.3 *Beachte, daß zu lange Programmzeilen nicht im ganzen auf dem Bildschirm sichtbar sind und von manchen Druckern abgeschnitten werden. Versuche deshalb, mit 80 Schreibspalten auszukommen.*