Allgemeine Hinweise zum Programmierstil

- *Richtigkeit und Zuverlässigkeit*

  Die Richtigkeit eines Programms lässt sich normalerweise nicht beweisen. Nur die *Anwesenheit* von Fehlern, nicht aber deren *Abwesenheit* kann nachgewiesen werden.

  - Bereits beim Entwurf berücksichtigen, dass Tests erforderlich sind.
  - Steuerparameter einbauen, die Tests veranlassen und an kritischen Stellen zusätzliche Ausgabe erzeugen.
  - Funktionen zuerst einzeln, dann in Gruppen und zuletzt im Gesamtprogramm testen.
  - Testbeispiele systematisch wählen: Vorkommende Fälle in Klassen einteilen und aus jeder Klasse mindestens ein Beispiel wählen (Äquivalenzklassenmethode). Evtl. zusätzlich Grenzfälle der Klassen untersuchen (Grenzwertanalyse).
  - Mögliche Schwachpunkte gezielt untersuchen.
  - Implizite Annahmen vermeiden, d.h. Voraussetzungen dokumentieren und zumindest im Testmodus überprüfen (z.B. mit assert).
  - Programmverhalten möglichst weitgehend spezifizieren, insbesondere mögliche Fehlersituationen erkennen und behandeln.
  - Erforderliche Initialisierungen von Konstanten und Variablen nicht vergessen.
  - Sorgfältig mit Zeigern umgehen, insbesondere Zugriffe auf nicht reservierten Speicherplatz vermeiden und die evtl. unterschiedliche Lebensdauer von Zeigervariablen und dynamisch reserviertem Speicherplatz beachten.

- *Effizienz*

  Verringerungen beim Rechenzeit- und Speicherplatzbedarf lassen sich oft nur durch einen Verlust an Übersichtlichkeit erreichen. Deshalb sollten Optimierungen auf die kritischen Stellen beschränkt werden.

  - Zuerst sicher programmieren, dann Optimierungen vornehmen.
  - Bessere Algorithmen wählen anstelle Geschwindigkeitserhöhung durch "Programmiertricks" zu erreichen versuchen.
  - Klar programmieren, einfache Optimierungen vom Compiler vornehmen lassen.
  - Bibliotheksfunktionen verwenden.
  - Laufzeitverhalten messen, dann gezielt die Stellen optimieren, an denen die meiste Rechenzeit verbraucht wird.
  - Speicherplatz dort einsparen, wo sehr viel verbraucht wird.
  - Bei Systemen mit virtueller Speichertechnik Seitenflattern (thrashing) vermeiden.

- *Bedienfreundlichkeit*

  Die Programmeingabe soll selbsterklärend, einfach und fehlertolerant sein.

  - Einzugebende Variablen durch Text anfordern.
  - Eingabewerte in das Ausgabeprotokoll schreiben.
  - Für die interaktive Eingabe nur formatfreie Leseanweisungen verwenden.
  - Daten interaktiv in kleinen Gruppen einlesen.
  - Nur wenige charakteristische Daten interaktiv, weitere Daten aus Dateien einlesen.
  - Den Inhalt von Eingabedateien mit Kommentar versehen, der vom Programm ignoriert wird.
  - Eingabewerte auf Plausibilität prüfen, gegebenenfalls die Datengruppe mit Erklärung neu anfordern und nicht sofort das Programm abbrechen.

Die Programmausgabe soll selbsterklärend, übersichtlich und ökonomisch sein.

- Nicht nur Zahlenwerte, sondern auch die Namen und evtl. Maßeinheiten der berechneten Größen ausgeben.
- Bei der Ausgabe von Zahlen nur die relevanten Stellen mit formatierten Schreibanweisungen drucken.
- Leerzeilen zur Strukturierung verwenden.
- Gegebenfalls Ergebnisse in Tabellenform drucken.
- Keine Datenfriedhöfe ausdrucken, sondern mit Steuerparametern den Umfang der Ausgabe bestimmen.

Bei der Benutzung durch andere eine aussagefähige Dokumentation schreiben und eine kurze Bedienungsanleitung in einer Dokumentationsdatei erstellen.

- *Wartungsfreundlichkeit und Portabilität*

  Änderungen an einem Programm und Tests sollen schnell und sicher vorgenommen werden können. Dazu ist insbesondere notwendig, dass sich der Programmtext leicht verstehen läßt.

  - Möglichst nur den standardisierten und maschinenunabhängigen Sprachumfang verwenden.
  - Einrückungen systematisch vornehmen.
  - Aussagekräftige Kommentare schreiben:
    - In einem Kommentar innerhalb des Programmtexts erklären, *weshalb* etwas gemacht wird und nicht nur die Anweisung einfach in Worten ausdrücken.
    - Ein Kommentar im Kopf einer Funktion sollte ihren Zweck, die Ein/Ausgabeparameter, die benutzten und evtl. geänderten globalen Variablen und Konstanten, die aufgerufenen Funktionen und die anzuschließenden Dateien enthalten. Außerdem ist es sinnvoll, das Erstellungsdatum und das Datum der letzten Änderung (bei größeren Projekten zusätzlich den Namen des Autors und die Versionsnummer) anzugeben.
    - Einen allgemeinen Funktionskopf auf einer Datei ablegen.
    - Bei Programmänderungen die Kommentare nicht zu ändern vergessen.
  - Variablen- und Funktionsnamen wählen, die über die Bedeutung (möglichst genau) Auskunft geben, insbesondere etwa Substantive für Variablen und Verben für Funktionsnamen verwenden.
  - Globale und lokale Namen unterscheiden, z.B. kurze Namen für lokale und ausführliche Namen für globale Variablen benutzen.
  - Temporäre Variablen sparsam verwenden.
  - Symbolische Konstanten anstelle “magischer“ Zahlen einsetzen.
  - Die Programmstruktur so festlegen, dass Gruppen von weitgehend voneinander unabhängigen Klassen und Funktionen entstehen, die sich leicht testen lassen.
  - Den Datenfluss möglichst einschränken, insbesondere möglichst wenige globale Variablen verwenden und nicht mehr Parameter übergeben als nötig.
  - Die Datenübergabe durch Parameter der durch globale Variable vorziehen. Das gilt insbesondere dann, wenn diese Werte verändert werden.
  - Unnötige Spezialisierungen vermeiden, d.h. Probleme allgemein lösen, wenn der Mehraufwand unerheblich ist.

Weitere Literatur: Kernighan, Pike: *The Practice of Programming.* Addison-Wesley 1999.