

Aufgabenstellung für das Programmierpraktikum 2003

1 Lernziele

Im Rahmen dieses Praktikums sollen Sie eine größere Programmieraufgabe mit einer vorgegebenen Programmiersprache (hier Pascal) lösen und dabei einen Eindruck davon erhalten, welche Anforderungen bzw. Probleme typischerweise im Umfeld der Softwareerstellung entstehen.

Es sollen die folgenden Lernziele erreicht werden:

1. Umsetzung einer nicht-trivialen Aufgabenstellung.
2. Selbständige Planung und Durchführung der Programmieraufgabe über einen längeren Zeitraum.
3. Entwicklung einer Lösung, die einer vorgegebenen Spezifikation für die Eingabe- und Ausgabedaten genügt.
4. Erzeugung einer Lösung, die ausreichend getestet und dokumentiert ist.

Das Praktikum ist so angelegt, dass solide Kenntnisse in Pascal bereits vorausgesetzt werden und nicht erst während des Praktikums erworben werden können.

2 Durchführung des Praktikums

Der Bearbeitungszeitraum für die Programmieraufgabe beginnt nach dem Versand der Aufgabenstellung am 04.04.2003; Ihnen steht also ein Zeitraum von fast fünf Monaten bis zum Abgabetermin am 01.09.2003 zur Verfügung. An diesem Termin erwarten wir von Ihnen das komplett funktionsfähige, getestete und dokumentierte Programm; diese Programmversion wird auch die Grundlage für die Vorführung und Erweiterung des Programmes in der Präsenzphase am 20.09.2003 oder 21.09.2003 sein. An welchem der beiden Termine Sie an der Präsenzphase teilnehmen können, teilen wir Ihnen per Email nach der Begutachten Ihrer Einsendungen mit. Sollten alle 90 Teilnehmer die Aufgabe erfolgreich lösen, müssen wir gegebenenfalls noch ein weiteres Wochenende für die Präsenzphase hinzunehmen.

Erfahrungsgemäß wird der Aufwand zum Erstellen und Testen der Programmlösung unterschätzt. Beginnen Sie daher möglichst gleich mit der Lösung der Aufgabe und planen Sie mindestens die gleiche Zeit zum Testen ein, die Sie für die Erstellung des Programms benötigt haben.

Später eintreffende Lösungen bzw. zwischen dem Abgabetermin und der Präsenzphase durchgeführte Programmänderungen werden bei der Bewertung nicht mehr berücksichtigt.

3 Wann erhalten Sie den Leistungsnachweis?

Dazu erwarten wir von Ihnen

1. Eigenständige Implementierung eines fehlerfreien Programmes gemäß der nachfolgenden Spezifikation.

Wichtig: Gruppenlösungen sind nicht zulässig.

2. Einsendung des Programms per Email bis zum **01.09.2003** unter Berücksichtigung der von uns angegebenen Programmier- und Dokumentationsrichtlinien.

Email-Adresse: daniela.keller@fernuni-hagen.de

Zu Ihrer Abgabe müssen die folgenden Komponenten gehören:

- 2.1 das Computerprogramm als .exe-Datei
 - 2.2 der Quelltext als .pas-Datei
 - 2.3 die Dokumentation als ASCII-, Word- oder PDF-Datei
3. Teilnahme an der Präsenzphase.

In der Präsenzphase stellen Sie Ihr Programm vor, präsentieren sein Verhalten anhand ausgesuchter Beispiele und erweitern Ihre Lösung um eine zusätzliche Funktionalität.

Manipulationsversuche mit den eingesandten Programmen, z.B. durch Aufspielen von Computerviren oder anderen Programmkomponenten, die nicht der Lösung der Programmieraufgabe dienen, führen - unabhängig von Schadensersatzansprüchen seitens der FernUniversität - zu einer Nichterteilung des Leistungsnachweises.

Der Leistungsnachweis für das Programmierpraktikum ist unbenotet.

4 Die Programmieraufgabe

Berechnungen zum Wetterbericht, zur Stabilität von Brücken, auf dem Gebiet der Regelungstechnik und der Analyse elektrischer Netzwerke führen zu sogenannten verallgemeinerten Eigenwertproblemen. Eine Möglichkeit zur Behandlung dieser Eigenwertprobleme ist ein Divide-et-Impera-Algorithmus ("Divide-and-Conquer"), der das Eigenwertproblem schließlich auf das Finden der Nullstellen einer speziellen rationalen Funktion zurückführt. Wichtig ist, die Nullstellen mit hinreichender Genauigkeit zu bestimmen, da davon die Genauigkeit, ja sogar die Durchführbarkeit des Verfahrens abhängt.

Bis heute wird die Sprache Pascal gerne für die Lösung mathematischer Probleme verwendet. Das Problem ist aber, dass sie das Rechnen mit beliebiger Genauigkeit nicht direkt unterstützt, sondern "nur" die Datentypen "*simple, real, comp, double, extended*" anbietet.

Die Aufgabe Ihres Programmierpraktikums besteht aus zwei Teilen:

- *Rechnen mit beliebig langen Zahlen*
- *Finden der Nullstellen einer speziellen rationalen Funktion*

4.1 Rechnen mit langen Zahlen

Schreiben Sie ein Programm, das

1. die vier Grundrechenarten sowie die Operationen größer, kleiner und gleich für beliebig lange ganze, Fest- und Gleitkommazahlen einer beliebigen Zahlenbasis realisiert.
2. fehlerhafte Eingaben abfängt.

Beachten Sie dabei:

1. Eine Zahl a wird bezüglich einer Basis b wie folgt dargestellt:

$$\begin{aligned} a &= (a_n a_{n-1} \dots a_1 a_0, a_{-1} \dots a_{-k})_b \\ &= a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b + a_0 + a_{-1} b^{-1} + \dots + a_{-k} b^{-k} \end{aligned}$$

mit $0 \leq a_i < b$.

Dabei bezeichnet n die Anzahl der Vorkommastellen und k die Anzahl der Nachkommastellen der Zahl a , wie auch im folgenden Beispiel deutlich wird:

$$(520,3)_6 = 5 \cdot 6^2 + 2 \cdot 6^1 + 0 + 3 \cdot 6^{-1}$$

Im Rahmen dieser Aufgabe gilt für die Basis b die folgende Einschränkung:

$$b \in B = \{2, \dots, 36\}$$

Ein Rechenausdruck enthält nur Zahlen bezüglich einer festen Basis. Die Ausgabe des Ergebnisses soll bezüglich der gleichen Basis erfolgen.

2. Aufgrund der Beschränkung der Basis auf die Menge B gilt:

$$\forall a_i \text{ mit } -k \leq i \leq n: a_i \in \{0, \dots, 35\}$$

Die a_i werden durch Zeichen aus der Menge $D = \{ '0', \dots, '9', 'A', \dots, 'Z' \}$ dargestellt.

Wir begrenzen die Anzahl der Nachkommastellen k .

Ihre Datenstruktur zur Darstellung einer beliebig langen Zahl darf aber keine fest definierten Grenzen aufweisen.

3. Bei der Ausgabe von Ergebnissen sollen führende bzw. anhängende Nullen unterdrückt werden, zum Beispiel sollte nicht 00001,00000, sondern 1 ausgegeben werden.

Ihr Programm sollte fehlerhafte Eingaben durch die Ausgabe einer entsprechenden Meldung an Stelle des Ergebnisses anzeigen, z.B.

- Zahlen bezüglich einer höheren Basis als der oben vorgegebenen,
- Division durch Null

4.2 Anwendung Nullstellenverfahren

Die im Teil 1 entwickelten Funktionen für Summe, Differenz, Produkt und Quotient sowie die Vergleichsoperatoren dienen als Grundlage, um die Nullstellen der rationalen Funktion

$$w(x) = 1 + \sum_{i=1}^q \frac{z_i^2}{\alpha_i - x} - \sum_{i=q+1}^m \frac{z_i^2}{\alpha_i - x}$$

zu berechnen.

Für diese Funktion w gilt:

- Die Koeffizienten z_i sind von Null verschiedene reelle Zahlen.
- Die reellen Polstellen α_i sind der Größe nach geordnet, alle Polstellen sind verschieden, d.h.

$$\alpha_1 < \alpha_2 < \dots < \alpha_q < \alpha_{q+1} < \dots < \alpha_{m-1} < \alpha_m,$$

- $m \geq 2$,
- $q \geq 1$,
- $q < m$.

Im allgemeinen ist es recht schwierig, die Nullstellen einer solchen Funktion zu bestimmen, da man meist gar nicht weiss, wieviele Nullstellen es gibt und in welchen Intervallen sie liegen. Doch solche Informationen sind wichtig für die Anwendung numerischer Nullstellenverfahren.

Bei einer Funktion w mit den oben beschriebenen Eigenschaften hat man die folgenden Informationen:

a. Für $j = 1, \dots, q$ gilt:

$$\lim_{x \rightarrow \alpha_j, x < \alpha_j} w(x) = \infty, \quad \lim_{x \rightarrow \alpha_j, x > \alpha_j} w(x) = -\infty,$$

d.h. die Funktion w hat in jedem der Intervalle $[\alpha_1, \alpha_2], \dots, [\alpha_{q-1}, \alpha_q]$ einen Vorzeichenwechsel und damit eine Nullstelle.

b. Im Intervall $[\alpha_q, \alpha_{q+1}]$ ist w eine konkave Funktion. Sie nimmt in diesem Intervall ein Maximum x_{\max} an und an den Intervallrändern geht sie gegen $-\infty$.

Wenn der Funktionswert zum Maximum x_{\max} positiv ist, hat w zwei Nullstellen in diesem Intervall. Ist $f(x_{\max}) = 0$, so liegt eine doppelte Nullstelle vor und bei $f(x_{\max}) < 0$ gibt es keine Nullstelle.

c. Für $j = q + 1, \dots, m$ liegt wegen

$$\lim_{x \rightarrow \alpha_j, x < \alpha_j} w(x) = -\infty, \quad \lim_{x \rightarrow \alpha_j, x > \alpha_j} w(x) = \infty,$$

in jedem der Intervalle $[\alpha_{q+1}, \alpha_{q+2}], \dots, [\alpha_{m-1}, \alpha_m]$ eine Nullstelle.

Verwenden Sie das im folgenden beschriebene Pegasus-Verfahren zur Berechnung der Nullstellen der Funktion w . Für die Operationen Addition, Subtraktion, Multiplikation, Division und die Vergleichsoperatoren gebrauchen Sie die von Ihnen in Teil 1 entwickelten Konstrukte, die Koeffizienten und Polstellen der Funktion w sowie alle anderen vorkommenden reellen Zahlen werden wie im Abschnitt „Rechnen mit langen Zahlen“ dargestellt, die Basis ist immer 10.

Im Fall a) und c) können Sie das Pegasus-Verfahren direkt anwenden (s. unten), im Fall b) haben Sie mehr Arbeit:

1. Berechnen Sie mit dem Pegasus-Verfahren die Nullstelle x_0 der Ableitung von w .
2. Berechnen Sie den Funktionswert für diese Nullstelle:
 Ist der Funktionswert größer Null, so suche man mit dem Pegasusverfahren eine Nullstelle im Intervall $[\alpha_q, x_0]$ und im Intervall $[x_0, \alpha_{q+1}]$.
 Ist der Funktionswert gleich Null, so hat man die Nullstelle bereits gefunden.
 Ist er kleiner Null, so gibt es in diesem Intervall keine Nullstelle.

4.2.1 Das Pegasus-Verfahren

Das Pegasus-Verfahren gehört zu den einschlusserhaltenden Verfahren. Ausgehend von zwei Startwerten x_0 und x_1 und den zugehörigen Funktionswerten $f_i := f(x_i)$ sollen x_i ($i = 2, 3, \dots$) so bestimmt werden, dass die jeweils letzten beiden Iterierten die Nullstelle einschließen, dass also die zugehörigen Funktionswerte unterschiedliche Vorzeichen haben.

Dazu wird zu je zwei Iterierten x_{i-1} und x_i die Gerade durch die Punkte (x_{i-1}, f_{i-1}) und (x_i, f_i) gelegt. Der Schnittpunkt dieser Geraden mit der x -Achse sei x_{i+1} . Es liegt zwischen x_{i-1} und x_i , weil f_{i-1} und f_i unterschiedliche Vorzeichen haben. Ist $f_{i+1} = f(x_{i+1}) = 0$, so hat man die Nullstelle gefunden und kann die Iteration abbrechen. Dieses Verfahren ist das **Sekantenverfahren**, welches aber nicht unbedingt die Nullstelle einschließt, da nicht garantiert ist, dass f_i und f_{i+1} verschiedene Vorzeichen haben.

Daher modifizieren wir es wie folgt: Liefern x_i und x_{i+1} keinen Einschluss, so überschreiben wir die Werte x_i und f_i durch x_{i-1} und $\gamma_i f_{i-1}$ mit dem Parameter $\gamma_i > 0$. So wird erzwungen, dass x_i und x_{i+1} einen Einschluss liefern.

Der Parameter γ_i beeinflusst, wo der Schnittpunkt der Geraden mit der x -Achse im nächsten Schritt liegt, und wirkt sich auf die Effizienz des Verfahrens aus.

Für $\gamma_i = 1$ erhält man die sogenannte **Regula Falsi**, die den Nachteil hat, dass man fast immer auf einer Seite mit einem x -Wert „hängen bleibt“ und sich nur der andere x -Wert der Lösung annähert.

Dieses Problem behebt das **Pegasus-Verfahren**, das mit $\gamma_i = \frac{f_i}{f_i + f_{i+1}}$ arbeitet.

Dieses Pegasus-Verfahren benutzen Sie zur Bestimmung der Nullstellen in den einzelnen Intervallen.

Da die Nullstellen, wenn sie existieren, zwischen den Polstellen der Funktion liegen, kann man je zwei benachbarte Polstellen als Startwerte für die Nullstellensuche wählen. Da aber der Funktionswert von w an diesen Stellen plus-unendlich oder minus-unendlich ist, muss man eine möglichst große positive Zahl mit dem entsprechenden Vorzeichen versehen. Damit die Berechnungen alle gleich beginnen, geben wir als große Zahl 10^{k-2} vor.

Die vorgegebene Genauigkeitsschranke ist $\varepsilon = 10^{-k+2}$, wobei k die Anzahl der Nachkommastellen darstellt. Diese Genauigkeitsschranke ist z.B. wichtig, um zu entscheiden, ob eine Zahl gleich null ist oder nicht. So gilt eine Nullstelle x_0 gilt als gefunden, wenn wir $|f(x_0)| < \varepsilon$, $\varepsilon = 10^{-k+2}$, erhalten.

4.2.2 Praktische Durchführung

Wir wissen, wo die Nullstellen der Funktion w zu suchen sind. Aber wir haben trotzdem keine Garantie, dass wir mit einem numerischen Verfahren alle Nullstellen finden. Das Pegasus-Verfahren ist ein wirklich gutes Verfahren zur Bestimmung der Nullstellen von Polynomen, aber die hier zu behandelnde rationale Funktion w ist sehr anfällig gegenüber Rundungsfehlern. Daher kann es passieren, dass man in einem oder sogar mehreren Intervallen keine Nullstelle findet, wenn man mit zu geringer Genauigkeit rechnet. Leider weiss man nicht, welches die „richtige“ Genauigkeit ist.

Fehler wegen zu geringer Rechengenauigkeit können an unterschiedlichen Stellen während der Nullstellensuche passieren. Denken Sie daher insbesondere daran, dass Sie vor jedem weiteren Iterationsschritt im Pegasus-Verfahren testen, ob noch ein Vorzeichenwechsel der Funktion in dem Intervall vorliegt. Wenn nicht, darf das Pegasus-Verfahren für das aktuelle Intervall hier nicht fortgesetzt werden und es muss eine Fehlermeldung ausgegeben werden.

Kritisch ist die Bestimmung der Startwerte für das Pegasus-Verfahren. Zwar hat die Funktion w bis auf das spezielle Intervall $[\alpha_q, \alpha_{q+1}]$ immer einen Vorzeichenwechsel in den Intervallen $[\alpha_i, \alpha_{i+1}]$, $i = 1, \dots, m$, $i \neq q$, aber der Funktionswert an den Polstellen ist nicht berechenbar. Daher muss man ein geeignetes kleines Stück weg von der Polstelle ins Intervall hineingehen und überprüfen, ob man für diese neuen Startwerte noch einen Vorzeichenwechsel hat.

Sorgen Sie auch dafür, dass Sie keine Endlosschleifen produzieren. Normalerweise sollten Sie nach 50 Iterationsschritten die Nullstelle in dem jeweiligen Startintervall gefunden haben oder aus anderen Gründen die Iteration beendet haben.

4.3 Eingabe und Ausgabe

4.3.1 Eingabe

Die beiden Aufgabenteile „Rechnen mit langen Zahlen“ und „Nullstellensuche“ werden von Ihnen innerhalb eines einzigen Programms realisiert. Dazu führen Sie zwei Modi ein, um zwischen den beiden Teilen „umschalten“ zu können. Bei Eingabe von **R** werden die geforderten Rechenoperationen für zwei Zahlen ausgeführt, bei Eingabe von **N** das Nullstellenverfahren für die rationale Funktion w gestartet.

Der Modus R:

Nach Eingabe von **R** geben Sie die Basis b , die Anzahl der Nachkommastellen k , die beiden Zahlen und den Namen der Operation ein, die ausgeführt werden soll. Jede Eingabe wird durch ein RETURN (Zeilenvorschub) abgeschlossen. Nach dem letzten RETURN beginnt Ihr Programm mit der Berechnung, gibt das Ergebnis aus und terminiert.

Die Zeichen der Operationen sind

- + für die Addition
- für die Subtraktion
- * für die Multiplikation
- / für die Division
- > für ist größer als
- < für ist kleiner als
- = für ist gleich

Beispiel: Bei der folgenden Eingabe

R
10
6
0,1
2,3
+

wird mit der Basis 10 und 6 Nachkommastellen $0,1+2,3$ berechnet.

Der Modus N:

Wählen Sie jetzt die feste Basis 10.

Nach der Eingabe des Modus **N** folgen die Eingabe der Nachkommastellenanzahl k und der Daten für die Funktion w :

N
Anzahl der Nachkommastellen
Anzahl der Polstellen m
Anzahl q
Polstelle α_1
...
Polstelle α_m
Koeffizient z_1
...
Koeffizient z_m

Achten Sie darauf, dass die Polstellen der Größe nach geordnet sind. Sollte dies nicht der Fall sein, muß eine entsprechende Fehlermeldung ausgegeben werden.

4.3.2 Ausgabe

Der Modus **R**:

Hier werden der Modus, Fehlermeldungen oder das Ergebnis der ausgeführten Operation ausgegeben.

Der Modus **N**:

Es werden die eingegebenen Daten ausgegeben, also

Modus Nullstellen: Eingabedaten
Anzahl der Nachkommastellen = k
Anzahl der Polstellen = m
Anzahl $q = q$
Polstelle α_1 = erste Polstelle
...
Polstelle α_m = m -te Polstelle
Koeffizient $z_1 = z_1$
...
Koeffizient $z_m = z_m$

Sobald ein Fehler bei der Eingabe festgestellt wird, bricht das Programm ab.

Nach erfolgreicher Eingabe werden dann die Ergebnisse der Nullstellensuche

Anzahl der gefundenen Nullstellen
Intervall [1.Polstelle, 2.Polstelle], Nullstelle = <Ausgabe der Nullstelle>
Intervall [2. Polstelle, 3. Polstelle], Nullstelle = <Ausgabe der Nullstelle>
...
Intervall [q.te Polstelle, q+1.te Polstelle]: eine Nullstelle gefunden: Nullstelle = <Ausgabe der Nullstelle> zwei Nullstellen gefunden: Nullstelle Nr.q = <Ausgabe der Nullstelle> Nullstelle Nr.q+1 = <Ausgabe der Nullstelle> keine Nullstelle gefunden
Intervall [q+2.te Polstelle, q+3.te Polstelle], Nullstelle = <Ausgabe der Nullstelle>
...
Intervall [m-1.te Polstelle, m.te Polstelle], Nullstelle = <Ausgabe der Nullstelle>

und eventuelle Fehlermeldungen des Programms ausgegeben.

Sollte Ihr Nullstellenprogramm in irgendeinem der Suchintervalle keine Nullstelle finden, so ist diese Meldung anstelle der gefundenen Nullstelle auszugeben, die Berechnung weiterer Nullstellen wird aber nicht abgebrochen.

5 Hinweise zur Turbo-Pascal

Vergewissern Sie sich, dass Ihre Lösung unter folgenden Rahmenbedingungen lauffähig ist:

- Benutzen Sie bitte nur Standard-Pascal-Konstrukte aus Turbo-Pascal 5.5, s. Email. Verwenden Sie insbesondere keine objektorientierten Pascal-Erweiterungen und keine über den Sprachumfang von Pascal hinausgehenden Programmbibliotheken. Es sei auch darauf hingewiesen, dass zur Erstellung der Lösung die Programmbibliothek *crt* nicht benötigt wird.

Desweiteren ist die Verwendung des *untypisierten* Zeigertyps „Pointer“ *nicht* erlaubt. Insbesondere ist es nicht erlaubt, die Prozeduren zur Verwaltung untypisierter Speicherstrukturen *getmem* und *freemem* zu benutzen.

- Die Entwicklung von graphisch „nett“ aussehenden **Benutzeroberflächen** ist ausdrücklich unerwünscht, da wir Ihre Lösung nur anhand der Korrektheit der produzierten Ausgabe bewerten werden.
- Ihr Programm darf keine speziellen DOS-Treiber bzw. -Konfigurationen zur Lauffähigkeit benötigen.

6 Dokumentationsrichtlinien

Neben dem eigentlichen Programm werden wir uns auch ausführlich mit Ihrer Dokumentation beschäftigen. Sie sollten daher bei der Gestaltung Ihrer Programmdokumentation die gleichen Maßstäbe ansetzen, nach denen Sie beispielsweise eine schriftliche Seminararbeit erstellen würden.

Halten Sie sich bei der Erstellung Ihrer Dokumentation bitte an die folgenden Richtlinien:

Ihre Dokumentation besteht aus einer **kurzen Einführung** und dem **kommentierten Programmcode**.

- In der **Einführung** (ca. eine DIN-A4-Seite) beschreiben Sie das generelle Vorgehen Ihres Programmes. Skizzieren Sie seine Grundidee, die wichtigsten benutzten Datenstrukturen und den grundlegenden Aufbau Ihres Programmes (z.B. welche Module erledigen welche Aufgaben). Gehen Sie aber an dieser Stelle noch nicht auf Implementierungsdetails ein; Ihre Einführung sollte beispielsweise auch von einem Leser nachvollziehbar sein, der zwar grundlegende Programmiererfahrungen besitzt, aber die Programmiersprache Pascal nicht beherrscht. Vermeiden Sie es, diesen Teil als großen Pascal-Kommentar vor Ihr Programm zu „quetschen“. Vergessen Sie auch nicht, Ihren Namen und Ihre Matrikelnummer auf dem Deckblatt anzugeben.
- Überlegen Sie sich einen **einheitlichen Kommentarkopf**, den Sie vor jeder Funktion und Prozedur einfügen. In diesem Kommentarkopf beschreiben Sie den Zweck der verwendeten Parameter, welche Aufgabe von der betreffenden Prozedur bzw. Funktion erfüllt wird, sowie die Einordnung der Prozedur/Funktion in den Gesamtkontext des Programmes. Vermeiden Sie es, in dem Kommentarkopf ganze „Romane“ zu schreiben; je zwei bis drei prägnante Sätze zur Aufgabe und Einordnung der Prozedur bzw. Funktion sagen mehr als eine halbe Seite Erläuterungstexte.
- **Kommentieren Sie Ihren Programmtext**. Das soll nicht heißen, dass Sie zu jeder Anweisung einen Kommentar schreiben müssen, aber Ihr Programm muss mit Hilfe der Kommentare soweit verständlich sein, dass der Leser Ihre Lösung ohne ein langwieriges Hineindenken in Ihre Pascal-Konstrukte nachvollziehen kann. Bedenken Sie dabei auch, dass der Leser im Gegensatz zu Ihnen nicht mit den von Ihnen eingeführten Datenstrukturen und Funktionen vertraut ist. Ersparen Sie ihm daher ein Nachblättern der betreffenden Datentypen oder Funktionen, indem Sie bei Wertzuweisungen stichwortartig erklären, was dort bezweckt/ausgeführt wird, wenn dies nicht offensichtlich ist.

Schließlich noch ein Hinweis: Wenn Sie bemerken, dass innerhalb einer Prozedur oder Funktion die Notwendigkeit auftritt, mehrere Sätze zur Erläuterung eines Programmabschnittes zu schreiben, dann ist dies ein Anzeichen dafür, dass Ihr Programm noch nicht genügend modularisiert ist. In diesem Fall sollten Sie erwägen, die betreffenden Programmteile als eigenständige Funktion auszugliedern. Auf keinen Fall dürfen Sie aber das Problem so „lösen“, dass Sie den betreffenden Teil nicht oder nur unzureichend kommentieren!

- Verwenden Sie **aussagekräftige Bezeichner** für Ihre Variablen, Funktionen und Prozeduren: Ein gut gewählter Bezeichner ist so kurz wie möglich, aber lang genug, um seine Funktion verständlich zu beschreiben. Abkürzungen sind erlaubt, sollten

aber „entschlüsselbar“ sein. Geben Sie Abkürzungen aus dem normalen Sprachgebrauch den Vorzug vor Eigenkonstrukten (also z.B. „Nachf“ für „Nachfolger“ und nicht „Nfolg“. Achten Sie auch darauf, dass Abkürzungen aussprechbar bleiben (z.B. „elem“ für „Element“ und nicht „elmt“)

- Benutzen Sie ein **einheitliches Vorgehen bei der Gliederung von Deklarationen und der Einrückung von Programmteilen**. Anweisungsfolgen zwischen **begin** und **end** werden mit 2 bis 4 Leerzeichen eingerückt; die Schlüsselworte **begin** und **end** stehen auf der gleichen Ebene wie die übergeordneten Konstrukte, zu denen sie gehören. Also zum Beispiel:

```
if <Bedingung> then
begin
  <Anweisungsfolge>
end
```

Aber **nicht**:

```
if <Bedingung> then
  begin
    <Anweisungsfolge>
  end
```

Das zusätzliche Einrücken von **begin** und **end** ist eine Unsitte, die sich leider in einigen Lehrbüchern findet; sie führt bei komplexeren Programmen jedoch lediglich zur Unübersichtlichkeit.

- Schreiben Sie **Pascal-Schlüsselworte** in Kleinbuchstaben wie in dem obigen Beispiel. Die in einigen Texten noch zu findende Schreibweise mit Großbuchstaben für Schlüsselworte ist veraltet und der Lesbarkeit des Programmes abträglich.
- Vermeiden Sie es, mehr als eine Anweisung in eine Zeile zu schreiben.

7 Hinweise zum Testen des Programms

Das Testen von Programmen ist ein sehr umfangreiches Fachgebiet innerhalb der Informatik, das Stoff genug enthält, um eine ganze Serie von Vorlesungen oder Kurseinheiten zu füllen.

Wir wollen Ihnen hier nur einige Denkanstöße aus diesem Gebiet liefern, um Ihnen das Testen und damit das Abliefern einer (fast, s.u.) korrekten Lösung zu erleichtern.

- a. Untersuchungen haben ergeben, dass ein Programm in der Realität niemals fehlerfrei ist. Für ein „frisch programmiertes“ Programm (also vor der Testphase) ist es realistisch anzunehmen, dass auf 100 Zeilen Code ca. 4 bis 8 Fehler kommen!

- b. Für nicht triviale Programme ist es aus Komplexitätsgründen nicht möglich, einen lückenlosen Korrektheitsbeweis zu führen, d.h. es ist für jedes reale Programm effektiv nicht beweisbar, dass es korrekt ist.

Daraus folgt, dass der Sinn des Testens eines Programms nicht darin bestehen kann, die völlige Fehlerfreiheit eines Programms nachzuweisen, da dies nach (a) extrem unwahrscheinlich und nach (b) objektiv ohnehin nicht beweisbar ist.

Daher definiert man das Testen häufig wie folgt:

„Testen bedeutet, ein Programm mit der Absicht auszuführen, Fehler zu finden.“

Eine Testeingabe wird als erfolgreich bezeichnet, wenn sie das Programm zu falschem Verhalten verleitet (fehlerhafte Ausgabe, Programmabbruch etc.). Hingegen betrachtet man eine Testeingabe als nicht erfolgreich, wenn sich das Programm korrekt verhält (korrekte Ausgabe oder Zurückweisung einer Eingabe, die außerhalb des gültigen Bereiches liegt).

Die Teststrategie besteht also darin, gezielt möglichst viele Fehler in dem Programm zu finden. Damit kann man zwar nicht beweisen, dass ein Programm überhaupt keine Fehler mehr enthält (s.o.), das Vertrauen in die Zuverlässigkeit des Programmes wird jedoch mit der steigenden Anzahl nicht erfolgreicher Testfälle (= korrekter Reaktionen des Programmes) und daraufhin eliminiertes Fehler erhöht.

Nachfolgend wollen wir Ihnen einige Anregungen geben, wie Sie Ihr Programm effektiv testen können:

- Zunächst einmal: Lassen Sie sich nicht dazu verleiten, Programmfehler als persönliche Fehlleistung aufzufassen (nach dem oben Gesagten sollte klar sein, dass sich Fehler zwangsläufig und unvermeidbar in Programme einschleichen)! Im Testen unerfahrene Programmierer empfinden das Testen häufig als unangenehm, da jeder Fehler als Rückschlag bzw. „Versagen“ aufgefasst wird, und die Konsequenz ist häufig, dass entweder überhaupt nicht oder nur sehr halbherzig (mit korrekten, für das Programm harmlosen Testfällen) getestet wird. Sehen Sie die ganze Sache positiv: Jeder Fehler, den Sie finden und beheben, macht Ihr Programm ein Stück perfekter!
- Wählen Sie Ihre Testeingaben effizient aus. Ein geeignetes Verfahren ist z.B. die **Grenzwertanalyse**. Dabei ermitteln Sie zunächst für einen Eingabewert alle gültigen und ungültigen Werte, und wählen dann jeweils einen Repräsentanten aus, der gerade noch gültig ist („auf der Grenze liegt“) und je einen Repräsentanten, der knapp außerhalb der Grenze liegt und damit ungültig ist. Wenn Sie z.B. eine Funktion testen, die einen Text mit einer Länge von 1..40 Zeichen als Eingabe bekommt, würden Sie nach der Grenzwertmethode jeweils einen Text der Länge 1 und einen Text der Länge 40 als gültige Eingabe sowie Texte der Längen 0 bzw. 41 als ungültige Werte auswählen. Die ungültigen Werte nimmt man in die Testmenge auf, um zu sehen, ob das Programm auch auf fehlerhafte Eingaben sinnvoll reagiert (indem es z.B. eine Warnung ausgibt o.ä.). Falls solche Testfälle nicht vorgesehen werden, kann es vorkommen, dass zum Beispiel ein Programm in inneren Modulen später aus „unerklärlichen“ Gründen abstürzt (weil im Verlauf der Berechnung intern ein ungültiger Wert erzeugt wurde), oder bei bestimmten Eingaben nur unsinnige (weil undefinierte) Ausgaben erscheinen.

- Eine ähnliche Strategie besteht darin, nach **Spezialfällen** (neutrale Elemente, Definitionslücken wie die berühmte Division durch Null) Ausschau zu halten. Arbeitet ein Sortieralgorithmus z.B. auch korrekt, wenn die Liste der zu sortierenden Worte leer, einelementig oder bereits sortiert ist?
- Versuchen Sie, eine Menge von Testeingaben so zu wählen, dass insgesamt alle Programmteile in möglichst **allen Kombinationen** durchlaufen werden.

Beispiel:

Zum Testen der Abfrage

```
if (a=3) then
  if (b=4) then <Anweisungsblock>
    else <Anweisungsblock>
  else <Anweisungsblock>
```

sind zumindest die Wertekombinationen

a = 3, b = 4,

a = 3, b <> 4.

a <> 3, b beliebig,

in die Testmenge aufzunehmen.

- Testen Sie Ihr Programm **modulweise**. Dies bedeutet, dass Sie das zu testende Modul direkt mit passenden Testwerten aufrufen und die zurückgegebenen Werte überprüfen. Ein in das Gesamtprogramm integriertes Modul lässt sich nicht mehr zielgerichtet testen, da die Eingabe des Hauptprogrammes auf dem „Aufrufweg“ zum Zielmodul oft so stark transformiert wird, dass man für das Modul keine individuellen Testwerte mehr erzeugen kann. Gleiches gilt natürlich für die Ausgabe des Moduls, die bis zur endgültigen (Bildschirm-)ausgabe im kompletten Programm so weit umtransformiert werden kann, dass sie zu dem betreffenden Modul nicht mehr eindeutig in Beziehung gesetzt werden kann.

Die von uns in der Präsenzphase zur Vorführung Ihres Programmes ausgewählte Testeingabe wird unter anderem auch einige Grenzfälle enthalten, die nach den oben beschriebenen Methoden aufgebaut sind. Wir behalten uns vor, ein unter diesen Voraussetzungen extrem instabiles oder fehlerhaftes Programm als nicht ausreichend zurückzuweisen.

8 Sonstiges

Ziel des Praktikums ist, Ihre Informatikkenntnisse und Programmierfähigkeiten zu überprüfen, nicht Ihre Mathematikgrundlagen. Wenn Sie mathematische Probleme mit dem zweiten Teil des Praktikums, der Nullstellensuche, haben sollten, wenden Sie sich ruhig per Email oder Telefon an

Daniela Keller, Tel. 02331/987-2794, Email: daniela.keller@fernuni-hagen.de

Schauen Sie bitte regelmäßig (spätestens jedeWoche) in die Newsgroup zum Praktikum. Zum einen haben Ihre Kommilitoninnen und Kommilitonen eventuell auch Anregung für Ihre Arbeit, zum anderen kündigen wir dort an, wann wir Emails an Sie versenden. So werden Sie z.B. von uns Beispieldaten zum Testen Ihres Programmes bekommen.

Wir wünschen Ihnen viel Erfolg bei diesem Praktikum

Ihre Praktikumsbetreuer

Dominic Heutelbeck

Daniela Keller

Stephan Lukosch

Jörg Roth