Prüfung zum Mathematisch-technischen Assistenten
Dokumentation der praktischen Arbeit

Thorsten Dickhaus

Prüfungskennziffer: 113
Programmiersprache: ANSI-C

27. November bis 01. Dezember 2000

Forschungszentrum Jülich GmbH
Zentralabteilung Forschungsreaktoren

Inhaltsverzeichnis

1 Aufgabenstellung **7**

2 Aufgaben- und Datenanalyse **15**
2.1 Aufgabenanalyse 15
2.2 Datenanalyse 16
2.2.1 Text, der das Suchmuster beschreibt 16
2.2.2 Zu durchsuchender Text 17
2.2.3 Muster und Musterbestandteil 17
2.2.4 Lösung 17
2.3 Ablaufplan 18

3 Verfahrensbeschreibung **19**
3.1 Einlesen des Suchtextes und des zu durchsuchenden Textes . . . 19
3.2 Analyse des Suchmusters 19
3.3 Suchen im Text 19
3.4 Ausgabe 20
3.5 Hauptprogramm 20

4 Programmbeschreibung **21**
4.1 Modulplan 21
4.2 Header-Datei global_header.h 22
4.2.1 Zweck 22
4.2.2 Makrodefinitionen 22
4.2.3 Datentyp bool 22
4.2.4 Datenstruktur musterbestandteil 22
4.2.5 Datenstruktur muster 22
4.2.6 Datenstruktur loesung 23
4.3 Header-Datei myerrors.h 23
4.3.1 Zweck 23
4.3.2 Makrodefinitionen 23
4.4 Modul main.c 24
4.4.1 Zweck 24
4.4.2 Funktion fehlerbehandlung() 24
4.4.3 Funktion main() 25
4.5 Header-Datei ein_ausgabe.h 28
4.5.1 Zweck 28
4.5.2 Enthaltene Prototypen 28
4.6 Modul ein_ausgabe.c 28
4.6.1 Zweck 28
4.6.2 Funktion eingabe() 28
4.6.3 Funktion ausgabe() 31
4.6.4 Funktion fehlerausgabe() 31
4.7 Header-Datei verarbeitung.h 32
4.7.1 Zweck 32

4.7.2 Enthaltener Prototyp . . . 32
4.8 Modul verarbeitung.c . . . 32
4.8.1 Zweck . . . 32
4.8.2 Funktion zuweisen() . . . 32
4.8.3 Funktion komplement() . . . 33
4.8.4 Funktion lese_zk() . . . 34
4.8.5 Funktion musteranalyse() . . . 35
4.8.6 Funktion suche_in_text() . . . 38
4.8.7 Hauptunterprogramm suche() . . . 40

5 Benutzeranleitung 43
5.1 Entwicklungsumgebung . . . 43
5.2 Compilation und Aufruf . . . 43
5.2.1 Compilation . . . 43
5.2.2 Aufruf . . . 43
5.3 Fehlercodes . . . 44
5.4 Regeln für die Eingabedatei . . . 44

6 Diskussion der Testbeispiele 45
6.1 Beispiele aus der Aufgabenstellung . . . 45
6.1.1 Erstes Beispiel aus der Aufgabenstellung . . . 45
6.1.2 Weitere Beispiele aus der Aufgabenstellung . . . 48
6.2 Weitere Normal- und Extrembeispiele . . . 50
6.2.1 Beispiel mit Satzzeichen und Umlauten . . . 50
6.2.2 Maximalbeispiel bezüglich der Anzahl an Musterbestandteilen . . . 50
6.2.3 Extrembeispiel bezüglich der Länge des zu durchsuchenden Textes: . . . 51
6.3 Sonderbeispiele . . . 52
6.3.1 Sonderbeispiel mit mehr als 20 Musterbestandteilen . . . 52
6.3.2 Sonderbeispiel mit mehreren gleichen Zeichenklassen mit Wiederholung nacheinander . . . 53
6.3.3 Sonderbeispiel: keinmaliges Vorkommen eines Zeichens . . . 54
6.3.4 Sonderbeispiel mit leerem Text . . . 54
6.4 Fehlerbeispiele . . . 54
6.4.1 Fehlerbeispiel mit nicht erlaubtem Zeichen . . . 55
6.4.2 Fehlerbeispiel mit leerer Zeichenklasse . . . 55

A Änderungen zum Vorentwurf vom ersten Tag 57
A.1 Abweichungen vom Entwurf am ersten Tag . . . 57
A.2 Korrekturen zum Entwurf am ersten Tag . . . 57
A.3 Ergänzungen zum Entwurf am ersten Tag . . . 57

B Hilfsmittel 59

C Erklärung 61

D Source-Code ANSI-C **63**
D.1 Header-Datei global_header.h 63
D.2 Header-Datei myerrors.h 65
D.3 Modul main.c 67
D.4 Header-Datei ein_ausgabe.h 70
D.5 Modul ein_ausgabe.c 71
D.6 Header-Datei verarbeitung.h 76
D.7 Modul verarbeitung.c 77

1 Aufgabenstellung

Mustersuche in Texten

Die Suche nach einer (kleinen) Zeichenkette (*Suchmuster* oder *Muster*) in einer größeren Zeichenkette (*Text*) ist ein klassisches Problem in der Datenverarbeitung.
In zeitgemäßen Anwendungen ist es üblich, nicht nur nach festen Zeichenketten zu suchen, sondern im Suchmuster gewisse *Metazeichen* zuzulassen, mit denen reguläre Ausdrücke formuliert werden können.

Schreiben Sie ein Programm, mit dem in einem Text nach einem Muster gesucht wird, das durch einen regulären Ausdruck beschrieben wird.

Für diese Aufgabe werden folgende Vereinbarungen getroffen:

1. Das Alphabet der erlaubten Zeichen umfasst alle Groß/Kleinbuchstaben, Ziffern, Leerzeichen, den *Zeilenvorschub* (sollte unter Ihrem Programmiersystem auf Ihrem Rechner ein Zeilenvorschub durch mehrere Zeichen dargestellt werden, so sind diese mehreren Zeichen logisch als ein erlaubtes Zeichen zu behandeln!) sowie die Satzzeichen '.', ',', '!', '?', ':' und ';'.
   Im Text (in dem nach dem Suchmuster gesucht werden soll) dürfen nur Zeichen aus diesem Alphabet auftreten.

2. Die Menge der Metazeichen ist: '.', '*', '[', ']', '^' und '\'. (Man beachte: Die Schnittmenge zwischen Menge der erlaubten Zeichen und Metazeichen ist nicht leer!)
   Im Suchmuster dürfen alle erlaubten Zeichen sowie (in der unten beschriebenen Form und Bedeutung) Metazeichen auftreten.
   Ein Zeichen, welches kein Metazeichen ist, steht für sich selber.
   Die Bedeutung der Metazeichen ist wie folgt:

'.' steht für jedes beliebige erlaubte Zeichen.

'[...]' (*Zeichenklasse*) steht für jedes der innerhalb der Klammern angegebenen (erlaubten) Zeichen. (Innerhalb '...' dürfen nur erlaubte Zeichen stehen, wobei mindestens ein Zeichen angegeben sein muss. Erlaubte Zeichen, welche auch Metazeichen sind, werden nicht als Metazeichen interpretiert!) So steht '[`aeiou`]' für jeden kleinen Vokal '`a`' oder '`e`' oder '`i`' oder '`o`' oder '`u`' und '[`0123456789`]' steht für jede Ziffer.

'[^...]' (*Zeichenklasse*) Komplement der Zeichenklasse '[...]', also jedes erlaubte Zeichen außer den in '...' angegebenen Zeichen. (Auch hier dürfen innerhalb '...' nur erlaubte Zeichen auftreten, wobei wiederum mindestens

ein Zeichen angegeben sein muss - Metazeichen werden nicht als solche interpretiert!) So steht '[^aeiou]' für jedes erlaubte Zeichen außer den kleinen Vokalen!

'*' steht für die beliebige Wiederholung des/der voranstehenden Zeichens oder Zeichenklasse (kein-, ein- oder mehrmals).
'a*' steht z.B. für kein, ein oder mehrere hintereinanderstehende a's.
'[0123456789]*' steht für jede beliebige (auch leere) Ziffernfolge.
'.*' steht für jede beliebige (auch leere) Zeichenkette.

'\' hebt die Sonderbedeutung des nachfolgenden Metazeichens auf (sollte das nachfolgende Zeichen kein Metazeichen sein, kann das '\' ersatzlos fortgelassen werden). '\.' steht beispielsweise für das erlaubte Zeichen '.', der Punkt wird also nicht mehr als Metazeichen aufgefasst.

Suchmuster können aus mehreren, hintereinanderstehenden Bestandteilen zusammengesetzt sein. Die Bedeutung ist dann naheliegend - folgendes Suchmuster:

```
[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*
```

ist beispielsweise wie folgt zu verstehen:

- Im Text muss zunächst ein Vorzeichen auftreten (1.Bestandteil: Zeichenklasse '[+-]'),

- anschließend muss die Ziffer '0' auftreten (2.Bestandteil: einfaches Zeichen),

- es muss ein Dezimalpunkt folgen (3.Bestandteil: '\.', die Sonderbedeutung des Metazeichens '.' ist aufgehoben!),

- hinter dem Dezimalpunkt muss eine von '0' verschiedene Ziffer folgen (4.Bestandteil: Zeichenklasse '[123456789]'),

- anschließend können beliebig viele Ziffern (auch keine mehr) folgen (5.Bestandteil: Zeichenklasse mit Wiederholung '[0123456789]*').

- Es muss ein großes 'E' oder kleines 'e' folgen (6.Bestandteil: Zeichenklasse '[Ee]'),

- anschließend wieder ein Vorzeichen (7.Bestandteil: Zeichenklasse '[+-]'),

- anschließend eine beliebige Ziffer (8.Bestandteil: Zeichenklasse '[0123456789]')

- und schließlich eine beliebige Ziffernfolge (9.Bestandteil: Zeichenklasse mit Wiederholung '[0123456789]*').

Dieser reguläre Ausdruck repräsentiert somit eine (etwas eingeschränkte Form der) Darstellung einer normalisierten Gleitpunktzahl.
Dieses Suchmuster würde etwa auf folgende Zeichenketten zutreffen:

'+0.4135e-23' und '-0.5E+0'

nicht jedoch auf folgende:

'0.5E-6'	(fehlendes Vorzeichen)
'+1.234E+6'	(nicht normalisiert)
'+0.123e25'	(fehlendes Vorzeichen im Exponenten)

- Schreiben Sie ein Unterprogramm, welches
  - ein Suchmuster und einen Text als Argument übernimmt,
  - das Suchmuster daraufhin analysiert, ob es den obigen Konventionen entspricht,
  - im Text nach der ersten Zeichenkette sucht, auf welches das Suchmuster zutrifft,
  - als Ergebnis die Position und Länge dieses „Treffers“ an den Aufrufer zurückliefert (sollten an der entsprechenden Position mehrere unterschiedlich lange „Treffer“ vorliegen, so ist die Länge des längsten zurückzugeben) bzw. eine Kennung, an der abgelesen werden kann, dass kein „Treffer“ vorliegt.

- Schreiben Sie ein Hauptprogramm, welches die Funktionalität Ihres Unterprogramms demonstriert.

Bibliotheksfunktionen zur Suche nach regulären Ausdrücken, welche möglicherweise in Programmbibliotheken der von Ihnen gewählten Programmiersprache vorhanden sind, dürfen nicht verwendet werden.

Beim Suchmuster:
`[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*`
und dem Text:

```
Die Gravitationskonstante ist g = +0.667E-10 Meter hoch drei durch
Kilogramm mal Sekunde zum Quadrat.
```

sollte die Ausgabe des Programms etwa wie folgt aussehen:

```
Muster:

[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*

Text:

Die Gravitationskonstante ist g = +0.667E-10 Meter hoch drei durch
Kilogramm mal Sekunde zum Quadrat.

Treffer: Position 35, Laenge 10
```

(Die Nummerierung der Positionen beginne mit dem 1.)
Beim gleichen Suchmuster:

`[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*`
und dem Text:

```
Die Gravitationskonstante ist g = 0.667E-10 Meter hoch drei durch
Kilogramm mal Sekunde zum Quadrat.
```

sollte die Ausgabe des Programms dann etwa wie folgt aussehen:

```
Muster:

[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*

Text:

Die Gravitationskonstante ist g = 0.667E-10 Meter hoch drei durch
Kilogramm mal Sekunde zum Quadrat.
```

```
Kein Treffer!
```

(Aufgrund des fehlenden Vorzeichens passt das Muster nicht auf die angegebene Zahl!)

Möglicher Lösungsansatz für das Unterprogramm:
Am Beispiel des Suchmusters:
`[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*`

1. Analyse des Suchmusters.

   a) Das Suchmuster wird (von vorne nach hinten) in seine einzelnen Bestandteile zerlegt, wobei jeder Bestandteil ein einzelnes Zeichen ohne Wiederholung (etwa '`0`'), eine Zeichenklasse ohne Wiederholung (etwa '`[Ee]`'), ein Zeichen mit Wiederholung (etwa '`a*`') oder eine Zeichenklasse mit Wiederholung (etwa '`[0123456789]*`') repräsentiert. Die Bestandteile werden durchnummeriert. (Es kann davon ausgegangen werden, dass das Suchmuster höchstens 20 solcher Bestandteile enthält!)
   Das Beispielsuchmuster hat folgende 9 Bestandteile:

Nr.	Bestandteil	Erläuterung
1.	`[+-]`	Zeichenklasse ohne Wiederholung
2.	`0`	einzelnes Zeichen ohne Wiederholung
3.	`\.`	einzelnes Zeichen ohne Wiederholung
4.	`[123456789]`	Zeichenklasse ohne Wiederholung
5.	`[0123456789]*`	Zeichenklasse mit Wiederholung
6.	`[Ee]`	Zeichenklasse ohne Wiederholung
7.	`[+-]`	Zeichenklasse ohne Wiederholung
8.	`[0123456789]`	Zeichenklasse ohne Wiederholung
9.	`[0123456789]*`	Zeichenklasse mit Wiederholung

   b) Zu jedem Bestandteil (unabhängig von Wiederholungen) wird die Menge der „passenden“ erlaubten Zeichen festgehalten.
   Diese Menge kann aus einem Zeichen bestehen oder (bei den Zeichenklassen '`[...]`' oder '`[^...]`' bzw. dem Metazeichen '.') aus mehreren Zeichen.

   c) Zu jedem Bestandteil wird vermerkt, ob bei diesem Wiederholung vorliegt (abschließendes Zeichen '`*`') oder nicht.

   Die so ermittelte Information zum Beispielsuchmuster sieht wie folgt aus:

Nr.	„passende“ Zeichen	Wiederholung
1.	+-	nein
2.	0	nein
3.	.	nein
4.	123456789	nein
5.	0123456789	ja
6.	Ee	nein
7.	+-	nein
8.	0123456789	nein
9.	0123456789	ja

Diese Analyse des Suchmusters braucht pro Suchmuster nur einmal durchgeführt zu werden. Sollte im nächsten Aufruf des Unterprogramms in einem anderen Text nach dem gleichen Suchmuster gesucht werden, kann evtl. auf die vormalige Analyse des Suchmusters zurückgegriffen werden!

2. Suche im Text.

    a) Im Text werden von vorne nach hinten alle möglichen Positionen daraufhin überprüft, ob an dieser Position ein „Treffer“ beginnt (wie diese Überprüfung aussehen kann, wird im nächsten Punkt erläutert!). Beim ersten Treffer wird das Unterprogramm mit entsprechendem Ergebnis beendet.

    b) Um zu überprüfen, ob an einer gewissen Position im Text ein „Treffer“ beginnt, simuliert man einen endlichen Automaten, der zu jedem Bestandteil des Suchmusters einen korrespondierenden Zustand besitzt und darüberhinaus einen weiteren Zustand, den sogenannten „Endzustand“.
    Ein Zustand (der Endzustand ausgenommen), dessen korrespondierender Bestandteil des Suchmusters ein Bestandteil mit Wiederholung ist, heiße „Sternzustand“.

    In unserem Beispiel hat der Automat somit 10 Zustände, Zustand 1 bis 9 korrespondieren zu den Bestandteilen des Suchmusters, Zustand 10 ist der Endzustand. Zustände 5 und 9 sind Sternzustände.

    Zu Beginn „befindet“ sich der Automat in Zustand 1 und die aktuelle Position im Text ist die Stelle, die daraufhin untersucht werden soll, ob hier ein „Treffer“ beginnt.

    Mit einem *Back-Tracking-Algorithmus* kann man nun versuchen, alle Zustände der Reihe nach zu durchlaufen und den Endzustand zu erreichen:

- Liegt an der aktuellen Position im Text ein zum aktuellen Zustand „passendes“ Teilstück vor, wechselt der Automat in den nachfolgenden Zustand und die Position im Text ist entsprechend weiterzusetzen.

- Falls kein passendes Teilstück vorliegt, wechselt der Automat in den letzten (zurückliegenden) Sternzustand (und die zugehörige Position im Text) und sucht nach einem anderen, zu diesem Sternzustand passenden Teilstück. (Hierzu ist es erforderlich, sich zu jedem Zustand die Position des passenden Teilstückes zu merken!)

Kann so der Endzustand erreicht werden, liegt ein „Treffer“ vor, ansonsten nicht.

Am **ersten** Tag sind abzugeben:

1. Aufgabenanalyse

2. Verbale Beschreibung des Verfahrens (Algorithmen, logische Datenstrukturen)

3. Angabe der physikalischen Datenstrukturen (Realisierung in der gewählten Programmiersprache)

4. Programmkonzeption (Modulplan, Funktion und Schnittstellen der Module)

5. Detaillierte Beschreibung des Algorithmus in Form von Pseudocode, Ablaufdiagramm oder Nassi-Shneiderman-Diagramm

Insgesamt sind abzugeben:

1. Rahmen-, Unterprogramm und Testdateien in maschinenlesbarer Form (die Programme sowohl als Quellcode als auch in ausführbarer Form)

2. Dokumentation

3. Beschreibung, Begründung und Diskussion der angegebenen Beispiele und eine ausreichende Zahl von Beispielen, die sowohl die Normalfälle als auch auftretende Sonder- und Fehlerfälle abdecken.

2 Aufgaben- und Datenanalyse

2.1 Aufgabenanalyse

Es ist ein Programm zu schreiben, welches in einem Text nach einem Muster sucht. Der Text besteht aus (beliebig vielen) Zeichen. Dabei sind folgende Zeichen zulässig:

(i) die 29 Kleinbuchstaben (einschließlich 'ä' , 'ö' , 'ü')

(ii) die 29 Großbuchstaben (einschließlich 'Ä' , 'Ö' , 'Ü')

(iii) 2 Vorzeichen ('+' , '-')

(iv) 2 Whitespace-Zeichen (Blank und Zeilenvorschub)

(v) 10 Ziffern

(vi) 2 runde Klammern ('(' und ')')

(vii) das Gleichheitszeichen ('=')

(viii) 6 Satzzeichen ('.' , ',' , '!', '?' , ':' , ';')

Hieraus ergibt sich, daß es genau 81 erlaubte Zeichen für den zu durchsuchenden Text gibt. Das Suchmuster darf alle oben genannten Zeichen beinhalten. Zusätzlich dürfen im Suchmuster Metazeichen vorkommen, die als Platzhalter für andere Zeichen bzw. eine Menge anderer Zeichen dienen. Als Metazeichen sind die sechs Zeichen '`.`', '`*`', '`[`', '`]`', '`^`' und '\' erlaubt. Eine „Sonderrolle" nimmt also das Zeichen '`.`' (Punkt) ein. Es ist sowohl ein erlaubtes Zeichen im zu durchsuchenden Text als auch ein Metazeichen, welches im Suchmuster eine besondere Bedeutung annehmen kann.
Durch Kombination der erlaubten und der Metazeichen können im Suchmuster reguläre Ausdrücke gebildet werden, die es dann im Text zu suchen gilt. Die Bedeutung der Metazeichen ergibt sich wie folgt:

(i) Mit dem Zeichen '`.`' kann ein beliebiges der 81 erlaubten Zeichen repräsentiert werden.

(ii) In den eckigen Klammern '`[`' und '`]`' können ein oder mehrere erlaubte Zeichen stehen. Der Ausdruck „paßt" dann auf jedes in '`[`' und '`]`' eingeschlossene Zeichen. Der Punkt '`.`', welcher sowohl Metazeichen als auch erlaubtes Zeichen ist, wird bei Einschluß in '`[`' und '`]`' als erlaubtes Zeichen interpretiert, die unter (i) erwähnte Sonderbedeutung geht verloren. Die Menge der Zeichen in eckigen Klammern wird im Folgenden als „Zeichenklasse" bezeichnet.

(iii) Ist innerhalb der eckigen Klammern das erste Zeichen ein '`^`', so wird dieses als ein weiteres Metazeichen interpretiert und bedeutet, daß der Ausdruck auf jedes erlaubte Zeichen „paßt", welches nicht innerhalb der eckigen Klammern steht. (Komplement)

(iv) Das Metazeichen '*' steht für die ein-, kein- oder mehrmalige Wiederholung des direkt voranstehenden regulären Ausdrucks (des direkt voranstehenden erlaubten Zeichens oder der direkt voranstehenden Zeichenklasse).

(v) Das Metazeichen '\' maskiert das nachfolgende Metazeichen, d.h. das nachfolgende Zeichen verliert seine Sonderbedeutung, wenn es denn eine hat. Anwendung des Zeichens '\' auf ein erlaubtes Zeichen, das kein Metazeichen ist, hat keinen Effekt. Daher hat das Metazeichen '\' nur in Verbindung mit dem nachfolgenden Zeichen '.' eine sinnvolle Sonderbedeutung, da alle anderen erlaubten Zeichen nicht maskiert zu werden brauchen und eine Maskierung der anderen Metazeichen einen ungültigen Suchstring zur Folge hat.

Mehrere reguläre Ausdrücke, die hintereinander im Suchmuster vorkommen, bedeuten, daß die einzelnen repräsentierten erlaubten Zeichen sequentiell im Text hintereinander vorkommen müssen, um auf das Suchmuster zu passen.
Zu ermitteln und auszugeben ist die Position im Text, an welcher das Suchmuster zuerst gefunden wird (d.h., an welcher ein Textabschnitt beginnt, der auf den durch das Suchmuster repräsentierten regulären Ausdruck paßt) sowie die maximale Länge dieses „Treffers".

Das Programm insgesamt gliedert sich in vier Teilaufgaben:

1. Einlesen des Musters und des Textes
2. Analyse des Suchmusters bezüglich syntaktischer Korrektheit und Zerlegung des Suchmusters in seine einzelnen Bestandteile
3. Suchen des Musters im Text
4. Ausgabe der ersten Position im Text, an welcher das Muster gefunden wurde sowie der maximalen Länge des zugehörigen Textabschnittes.

2.2 Datenanalyse

2.2.1 Text, der das Suchmuster beschreibt

Da das Suchmuster laut Aufgabenstellung höchstens 20 Einzelbestandteile enthalten kann, ist seine Länge limitiert. Jeder Einzelbestandteil kann entweder ein erlaubtes Zeichen (Länge 1), ein maskiertes Zeichen (Länge 2) oder eine Zeichenklasse (mehrere in '[' und ']' eingeschlossene Zeichen) sein. Da es jedoch nur 81 erlaubte Zeichen gibt, kann eine Zeichenklasse nur 81+3=84 Zeichen lang sein, da nur die beiden Zeichen '[' und ']' sowie ggfs. das Zeichen '^' (für Komplementbildung) hinzukommen können (mehrere identische Zeichen in einer Zeichenklasse machen keinen Sinn, da die Menge sich hierdurch nicht vergrößert). Es ist dann noch das abschließende Zeichen '*' möglich. Somit kann eine Zeichenkette, die ein Muster beschreibt, höchstens 20×85 Zeichen lang sein und zu ihrer Speicherung kann ein statisches Characterfeld der erwähnten Maximallänge verwendet werden.

2.2.2 Zu durchsuchender Text

Für die Länge des zu durchsuchenden Textes sind in der Aufgabenstellung keine Limitierungen angegeben. Da aber auf die einzelnen Zeichen des Textes wahlfreier Zugriff bestehen muß (Sprünge im Backtracking-Algorithmus) ist es auch für den Textstring erforderlich, ihn in einen Character-Vektor einzulesen, damit für die weitere Bearbeitung per Index auf seine einzelnen Zeichen zugegriffen werden kann. Die Länge des einzulesenden Textes läst sich aber dadurch im Voraus bestimmen, daß nach Einlesen des Suchtextes die verbleibende Größe der Eingabedatei ermittelt wird. Sodann ist es möglich, dynamisch soviel vom Heap-Speicher des Rechners anzufordern wie nötig.

2.2.3 Muster und Musterbestandteil

Um in dem Textstring nach dem Suchmuster suchen zu können, empfiehlt es sich, den Suchtext zunächst einmal in einzelne, nicht mehr teilbare Bestandteile, die ihrerseits reguläre Ausdrücke sind, zu zerlegen. Für diese Einzelbestandteile gilt, daß sie nur einen der folgenden vier Typen von regulären Ausdrücken annehmen können:

1. Einzelnes Zeichen ohne Wiederholung
2. Einzelnes Zeichen mit Wiederholung
3. Zeichenklasse ohne Wiederholung
4. Zeichenklasse mit Wiederholung

Zur Speicherung eines solches Bestandteils wird eine Datenstruktur `musterbestandteil` verwendet. In ihr werden die Menge der für den Bestandteil passenden Zeichen sowie ein Vermerk, ob Wiederholung zulässig ist oder nicht, gespeichert. Ein ganzes Muster setzt sich dann aus bis zu 20 dieser Musterbestandteile zusammen.

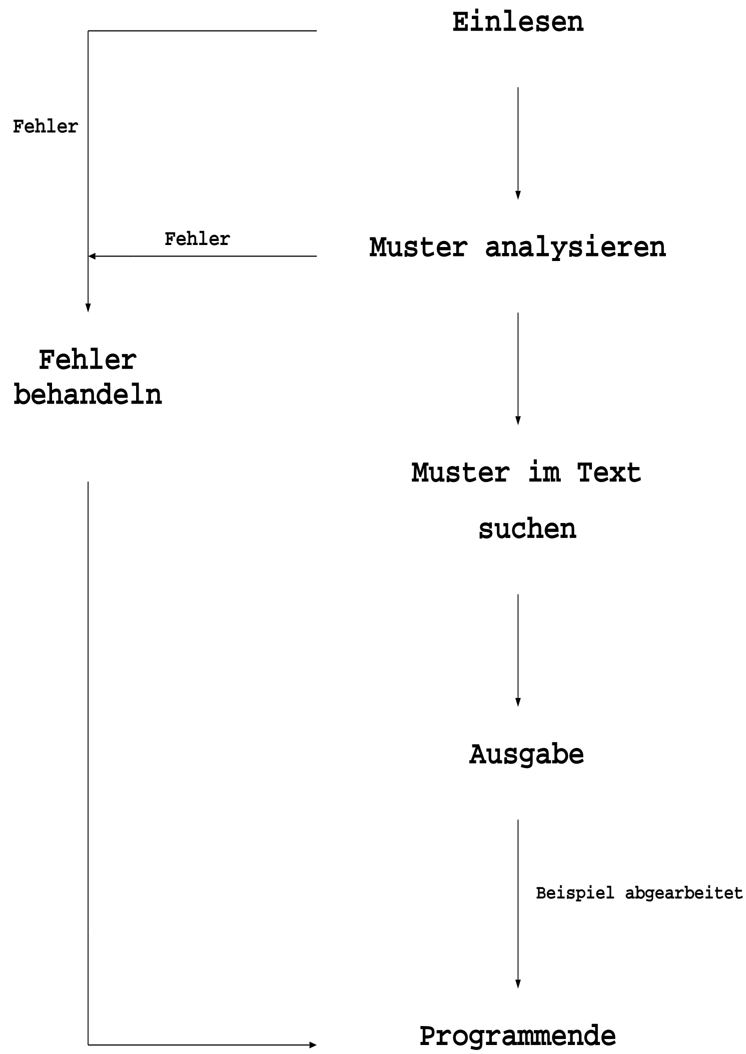
2.2.4 Lösung

Sollte das Muster im Textstring gefunden werden, so sollen zwei Angaben ausgegeben werden:

1. erste Anfangsposition
2. maximale Länge

Für diese Angaben wird eine Datenstruktur `loesung` benutzt, die die beiden Werte als Integer-Zahlen speichert.

2.3 Ablaufplan



3 Verfahrensbeschreibung

3.1 Einlesen des Suchtextes und des zu durchsuchenden Textes

Die Eingabe erfolgt grundsätzlich von Datei. In einer solchen Eingabedatei stehen sowohl der Suchtext (das Muster) als auch der zu durchsuchende Text. Zunächst wird der Suchtext Zeichen für Zeichen eingelesen und sofort für jedes Zeichen eine Gültigkeitsprüfung vorgenommen. Stellt sich das Zeichen als ungültig heraus, kann der Einlesevorgang mit einer entsprechenden Fehlermeldung abgebrochen werden, anderenfalls wird das Zeichen in den Zeichen-Vektor, der den Suchtext repräsentiert, übertragen.
Mit dem zu durchsuchenden Text wird genauso verfahren. Allerdings muß zuvor seine Größe unter Zuhilfenahme der Dateigröße ermittelt und entsprechend viel Speicherplatz dynamisch alloziert werden.

3.2 Analyse des Suchmusters

Hierbei halte ich mich an die in der Aufgabenstellung vorgeschlagene Verfahrensweise. Der Suchstring wird in einzelne Bestandteile aufgespalten, die entweder „Zeichen ohne Wiederholung“ oder „Zeichen mit Wiederholung“ oder „Zeichenklasse ohne Wiederholung“ oder „Zeichenklasse mit Wiederholung“ sein können. Die Funktion zur Analyse eines Suchmusters bekommt einen Character-String übergeben und liefert ein Resultat vom Typ `muster` wie in Kapitel 2 beschrieben.

3.3 Suchen im Text

Zur Suche des vorher analysierten Musters im eingelesenen Text wird ein Backtracking-Algorithmus verwendet, der einen endlichen Automaten simuliert. Die Zustände dieses Automaten entsprechen den in der Musteranalyse ermittelten Einzelbestandteilen des Suchmusters. Zudem gibt es noch einen Endzustand, der angibt, daß das Muster komplett im Text gefunden worden ist.
Es wird an einer Position im Text ermittelt, ob das Zeichen im Text zu dem ersten Musterbestandteil paßt. Ist dies der Fall, so wechselt der Automat in den nächsten Zustand (d.h., es erfolgt die Untersuchung, ob das folgende Zeichen zu dem zweiten Musterbestandteil paßt). So verfährt man weiter, bis entweder alle Musterbestandteile abgearbeitet worden sind (d.h., der Automat in den Endzustand übergehen kann) oder ein „Mismatch“ auftritt (Zeichen kann nicht dem Musterbestandteil zugeordnet werden). Bei einem solchen Mismatch geht der Algorithmus bis zum letzten Musterbestandteil mit Wiederholung zurück (der Automat wechselt in den vorangegangenen „Sternzustand“) und setzt seine Betrachtung an dieser Stelle fort. Daraus ergibt sich die Notwendigkeit, zu den verschiedenen Zuständen die zugehörigen Textposition festzuhalten, an welchen sie erkannt wurden, damit gegebenenfalls an dieser Position weitergesucht werden kann, falls nicht bis zum Endzustand verzweigt wird.

In der obersten Rekursionsstufe (im Endzustand) muß zudem noch die Länge des erkannten Musters festgehalten werden, damit das längste mögliche Muster an der betreffenden Stelle gefunden wird.
Darüber hinaus ist eine Funktion erforderlich, die diesen Backtracking-Algorithmus für alle Anfangspositionen aufruft, bis daß ein „Treffer“ gefunden wurde oder eine ungültige Anfangsposition erreicht wurde.

3.4 Ausgabe

Die Ausgabe des Programmes besteht darin, sowohl die Anfangsposition des ersten Textabschnittes, der auf das Muster paßt, sowie die maximale Länge dieses Textabschnittes auszugeben. Die Ausgabe kann entweder auf eine Datei oder auf die Standardausgabe (benutzerdefiniert) erfolgen.

3.5 Hauptprogramm

Das Hauptprogramm hat folgende Aufgaben:

1) Öffnen des Eingabe- und ggfs. des Ausgabefiles

2) Aufruf der Einleseroutine

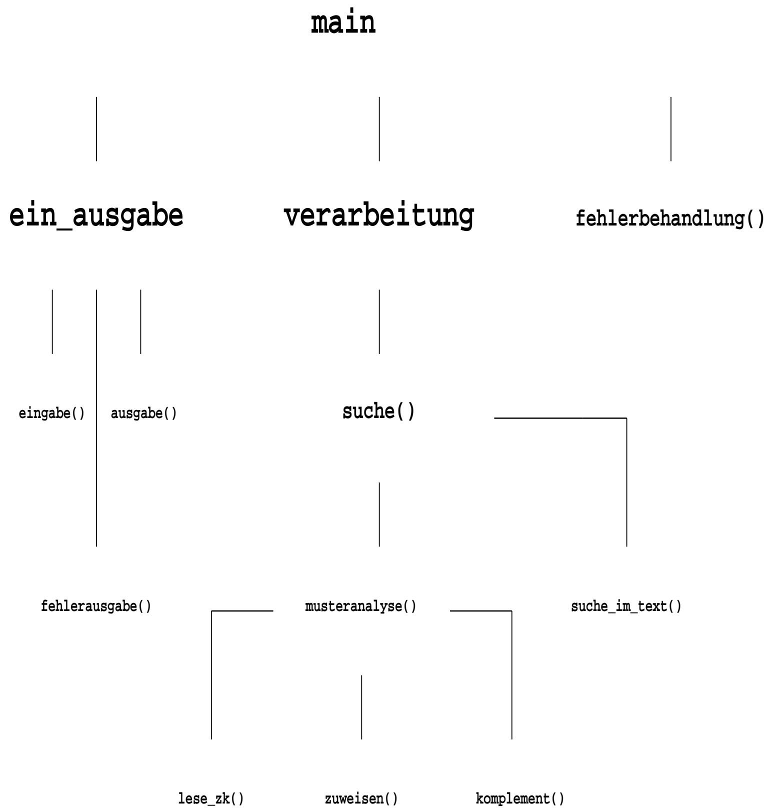
3) Aufruf des Unterprogramms zur Textsuche, welches die Ausgabedaten erzeugt.

4) Aufruf der Ausgaberoutine

Zwischen diesen Arbeitsschritten muß jeweils eine Fehlerüberprüfung erfolgen, und auf Fehler richtig (angemessen) reagiert werden.

4 Programmbeschreibung

4.1 Modulplan



4.2 Header-Datei global_header.h

4.2.1 Zweck

In der Header-Datei `global_header.h` befinden sich allgemein gültige Makrodefinitionen sowie die selbstdefinierten Datenstrukturen `bool`, `musterbestandteil`, `muster` und `loesung` für dieses Programm. Sie werden hier an zentraler Stelle vereinbart, damit Änderungen nur an einer Stelle vorgenommen werden müssen. Ferner werden die nötigen System-Headerdateien eingebunden. Die Header-Datei `global_header.h` muß somit in alle Module einbezogen werden.

4.2.2 Makrodefinitionen

Makroname	**Definition**	**Bedeutung**
`ALPHA`	81	Anzahl der Zeichen im Alphabet
`MAXBESTAND`	20	Maximalanzahl an Bestandteilen pro Muster

Beide Makros sind variabel, mit ihnen kann die Problemgröße gesteuert werden.

4.2.3 Datentyp bool

```
typedef enum {false, true} bool;
```

Mit diesem selbstdefinierten Aufzählungstyp wird ein logischer Datentyp modelliert, da es einen solchen in der Programmiersprache C nicht gibt. Dies erhöht die Lesbarkeit des Programms, denn Konstantennamen wie `true` und `false` sind leichter zu verstehen als die Konstanten `1` und `0`, die standardmässig in C zur Repräsentation von logischen Zuständen verwendet werden.

4.2.4 Datenstruktur musterbestandteil

```
typedef struct
{
 char passende_zeichen[ALPHA+1];
 bool wiederholung;
} musterbestandteil;
```

`musterbestandteil` repräsentiert einen regulären Teilausdruck, aus dem sich Suchmuster zusammensetzen. Es werden, wie in der Aufgabenstellung vorgeschlagen, die zu dem Ausdruck „passenden“ Zeichen (in einer Zeichenkette) sowie die Information, ob Wiederholung vorliegen darf oder nicht, abgespeichert.

4.2.5 Datenstruktur muster

```
typedef struct
{
 int bestandteile;
```

```
 musterbestandteil teile[MAXBESTAND];
} muster;
```

`muster` dient zur Speicherung eines ganzen Musters. Ein solches Muster setzt sich laut Aufgabenstellung aus höchstens 20 Einzelbestandteilen vom Typ `musterbestandteil` zusammen, weshalb ein statisches Feld dieses Typs der Maximallänge `MAXBESTAND` (hier 20) verwendet wird. Durch Änderung des Makros `MAXBESTAND` in der Header-Datei `global_header.h` ist es möglich, hier auch längere Muster zuzulassen oder aber die Maximallänge eines Musters weiter einzuschränken. Die Komponente `bestandteile` gibt an, wieviele Bestandteile das Muster tatsächlich hat.

4.2.6 Datenstruktur loesung

```
typedef struct
{
 int laenge;
 int anfang;
} loesung;
```

Mit `loesung` wird eine Datenstruktur definiert, die die zwei Komponenten beinhaltet, die im Falle einer erfolgreichen Suche nach dem Muster im Text ausgegeben werden sollen. `laenge` enthält die Maximallänge des gefundenen Textabschnittes, `anfang` seine Anfangsposition relativ zum Beginn des Textes (Zählung beginnt bei 1).

4.3 Header-Datei myerrors.h

4.3.1 Zweck

Die Header-Datei `myerrors.h` ist dazu da, um die im Programm verwendeten Fehlercodes durch Makros zu beschreiben. Dies erhöht die Lesbarkeit des Programms und erleichtert die Implementation der Fehlerbehandlungsroutine, da der übergebene Fehlercode sofort zu entschlüsseln ist.

4.3.2 Makrodefinitionen

Makroname	Definition	Bedeutung
`AUFRUF`	(-15)	Falscher Programmaufruf
`INDAT`	(-14)	Öffnen der Eingabedatei fehlgeschlagen
`OUTDAT`	(-13)	Öffnen der Ausgabedatei fehlgeschlagen
`SPEICHER`	(-12)	Nicht genügend Speicher zum Allozieren von text
`DAT_LEER`	(-11)	Eingabedatei enthält keine verwertbaren Daten

ZUENDE	(-10)	Unerwartetes Ende der Eingabedatei
NO_COMMENT	(-9)	Kommentarzeile fehlt in der Eingabedatei
NICHT_GETRENNT	(-8)	Separator zwischen Muster und Text fehlt
CHAR_SUCHTEXT	(-7)	Ungültige Zeichen im Suchtext
CHAR_TEXT	(-6)	Ungültige Zeichen im Text
SYNTAX_SUCHSTR	(-5)	Syntax im Suchstring ist inkorrekt
FALSCH_IN_ZK	(-4)	Unerlaubte Zeichen in einer Zeichenklasse
DOPPELT_IN_ZK	(-3)	Durch doppelte Zeichen in einer Zeichenklasse wird der Suchstring zu lang
LEERE_ZK	(-2)	Leere Zeichenklasse aufgetreten
NICHT_GEFUNDEN	(-1)	Muster kann nicht im Text gefunden werden

Alle Makros sind fest. Die Fehler sind geordnet (je schwerwiegender der Fehler, desto niedriger der zugehörige Fehlercode).

4.4 Modul main.c

4.4.1 Zweck

Das Modul `main.c` beinhaltet das Hauptprogramm, welches den Programmablauf und die Ein-/Ausgabe steuert. Ferner ist die Fehlerbehandlungsroutine `fehlerbehandlung()` enthalten.

4.4.2 Funktion fehlerbehandlung()

- Prototyp

```
void fehlerbehandlung(int fehler, FILE *in, int zaehler,
                      FILE *out, char *text);
```

- Eingabeparameter

Name	Datentyp	Bedeutung
fehler	int	Art des zu behandelnden Fehlers
in	FILE*	Zeiger auf die Eingabedatei
zaehler	int	Anzahl der an das Programm übergebenen Argumente

out	FILE*	Zeiger auf die Ausgabedatei
text	char*	Zeiger auf den dynamisch angelegten Speicher für den Text

- Ausgabeparameter
  keine
- lokale Variablen
  keine
- Rückgabewert
  keiner
- Beschreibung
  Alle angeforderten Systemressourcen werden freigegeben.

4.4.3 Funktion main()

- Prototyp

```
int main(int zaehler, char **argumente);
```

- Eingabeparameter

Name	**Datentyp**	**Bedeutung**
zaehler	int	Anzahl der an das Programm übergebenen Argumente
argumente	char**	Zeichenketten-Vektor mit den Argumenten

- Ausgabeparameter
  keine
- lokale Variablen

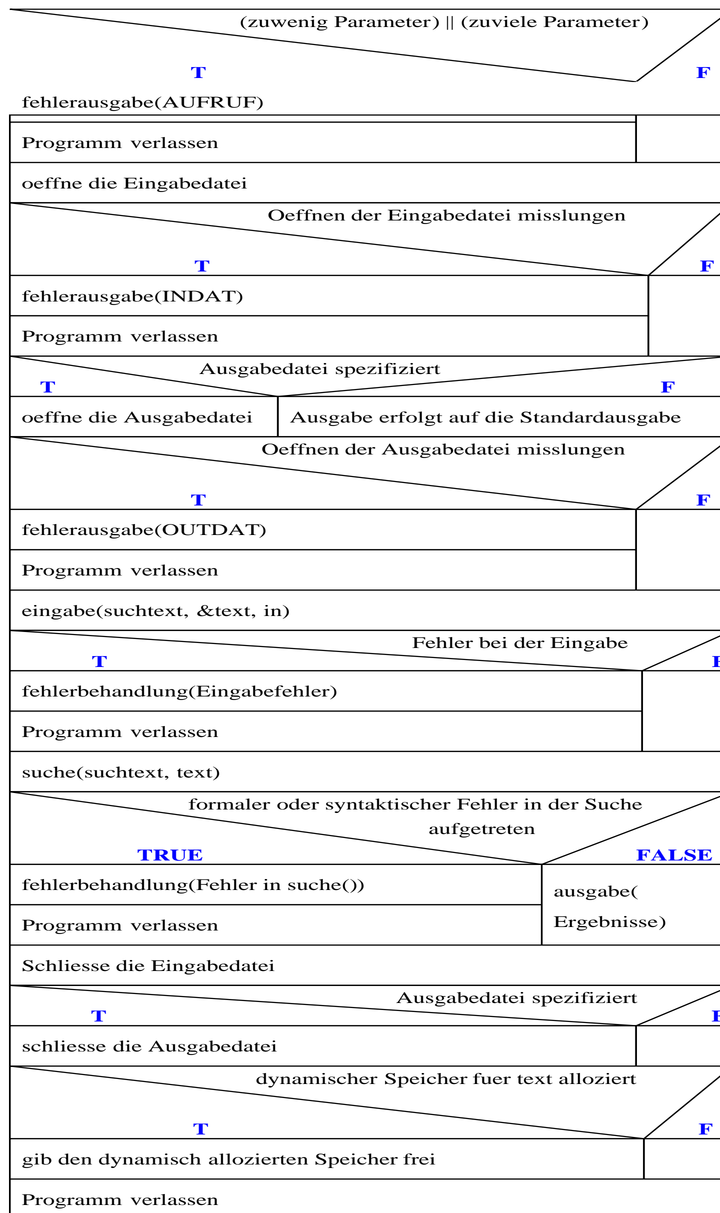
Name	**Datentyp**	**Bedeutung**
suchtext	char[]	Text, der das Muster beschreibt
text	char*	zu durchsuchender Text
in	FILE*	Zeiger auf die Eingabedatei
out	FILE*	Zeiger auf die Ausgabedatei
fc	int	Fehlercode
l	loesung	Ergebnisse der Textsuche

- Rückgabewert

Name	**Datentyp**	**Bedeutung**
keiner	int	Rückgabewert an die Aufrufumgebung

- Beschreibung
  Für ein Eingabebeispiel (Suchmuster und Text) wird die Textsuche durchgeführt. Treten Fehler auf, die wird die Fehlerbehandlungsroutine aufgerufen. Wenn nicht, dann erfolgt die Ausgabe mit Hilfe des Aufrufs der Funktion `ausgabe()`.

- Nassi-Shneidermann-Diagramm



4.5 Header-Datei ein_ausgabe.h

4.5.1 Zweck

Die Header-Datei `ein_ausgabe.h` stellt dem Hauptprogramm die Schnittstellen zu den benötigten Ein-/Ausgabefunktionen zur Verfügung.

4.5.2 Enthaltene Prototypen

```
extern int eingabe(char *suchtext, char **text, FILE *in);
extern int ausgabe(const char *suchtext, const char *text, loesung l,
                   FILE *out);
extern void fehlerausgabe(int fehler);
```

4.6 Modul ein_ausgabe.c

4.6.1 Zweck

In dem Modul `ein_ausgabe.c` sind alle für das Programm nötigen Ein-/Ausgaberoutinen implementiert.

4.6.2 Funktion eingabe()

- Prototyp

```
int eingabe(char *suchtext, char **text, FILE *in);
```

- Eingabeparameter

Name	**Datentyp**	**Bedeutung**
in	FILE*	Zeiger auf die Eingabedatei

- Ausgabeparameter

Name	**Datentyp**	**Bedeutung**
suchtext	char*	Zeichenkette, auf die der Suchtext eingelesen wird
text	char**	Zeiger auf die Zeichenkette, auf die der Text eingelesen wird (dynamischer Speicher)

- lokale Variablen

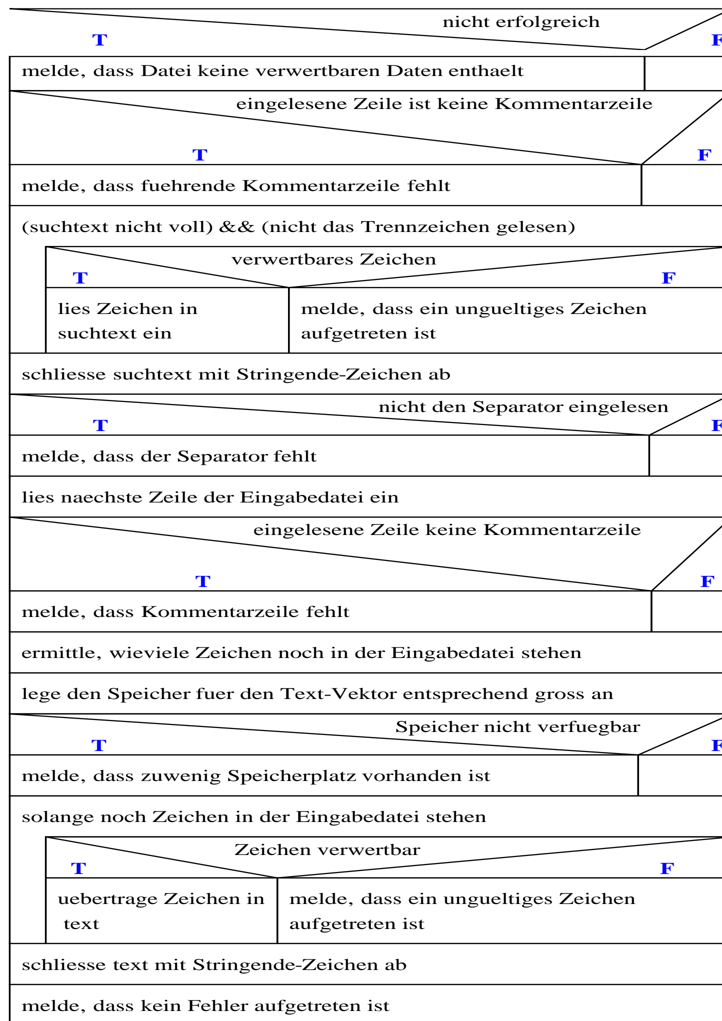
Name	Datentyp	Bedeutung
erlaubte_zeichen	const char[]	Vektor mit dem Alphabet
metazeichen	const char[]	Vektor mit den Metazeichen
dummy	char char[]	Zwischenspeicher
separator	char	Trennzeichen
c	int	Zwischenspeicher
i	int	Zählvariable
anzahl	int	Länge des Textes
aktuelle_position	long	Hilfsgröße für `anzahl`
endposition	long	Hilfsgröße für `anzahl`

- Rückgabewert

Name	Datentyp	Bedeutung
fc	int	Fehlercode

- Beschreibung
  Zuerst wird eine Kommentarzeile eingelesen, dann der Suchtext. Danach muß ein Separator eingelesen werden, der Suchtext und Text voneinander trennt. Nach einer weiteren Kommentar- Zeile wird mit Hilfe der Dateiposition die Länge des Textes bestimmt und der Text danach eingelesen. Zwischen allen Arbeitsschritten erfolgt eine Fehlerprüfung.

- Nassi-Shneidermann-Diagramm



4.6.3 Funktion ausgabe()

- Prototyp

```
int ausgabe(const char *suchtext, const char *text,
            loesung l, FILE *out);
```

- Eingabeparameter

Name	Datentyp	Bedeutung
suchtext	const char*	Text mit dem Suchmuster
text	const char*	zu durchsuchender Text
l	loesung	Ergebnisse der Textsuche
out	FILE*	Zeiger auf die Ausgabedatei

- Ausgabeparameter
  keine
- lokale Variablen
  keine
- Rückgabewert

Name	Datentyp	Bedeutung
keiner	int	Fehlercode

- Beschreibung
  Wie in der Aufgabenstellung gezeigt, werden das Suchmuster und der zu durchsuchende Text ausgegeben. Ferner werden die Ergebnisse der Textsuche ausgegeben. Die Ausgabe erfolgt entweder in die Ausgabedatei oder, wenn diese nicht spezifiziert wurde, auf die Standardausgabe.

4.6.4 Funktion fehlerausgabe()

- Prototyp

```
void fehlerausgabe(int fehler);
```

- Eingabeparameter

Name	Datentyp	Bedeutung
fehler	int	Art des Fehlers

- Ausgabeparameter
  keine
- lokale Variablen
  keine
- Rückgabewert
  keiner
- Beschreibung
  Zu einem Fehler wird eine entsprechende Fehlermeldung auf die Standard-Fehlerausgabe geschrieben.

4.7 Header-Datei verarbeitung.h

4.7.1 Zweck

Die Header-Datei `verarbeitung.h` stellt dem Hauptprogramm die Schnittstelle zu dem Haupt-Untergrogramm `suche()` zur Verfügung und sorgt mittels Einbinden der Datei `global_header.h` dafür, daß den Funktionen im Modul `verarbeitung.c` die globalen Datenstrukturen bekannt sind.

4.7.2 Enthaltener Prototyp

```
extern loesung suche(const char *suchtext, const char *text);
```

4.8 Modul verarbeitung.c

4.8.1 Zweck

Das Modul `verarbeitung.c` enthält die Funktionen zur Textsuche. Insbesondere ist in ihm das Haupt-Unterprogramm `suche()` enthalten.

4.8.2 Funktion zuweisen()

- Prototyp

```
static bool zuweisen(muster *m, const char *c, char stern);
```

- Eingabeparameter

Name	Datentyp	Bedeutung
c	const char*	zuzuweisender String
stern	char	nächstes Zeichen

- Ausgabeparameter

Name	Datentyp	Bedeutung
m	muster*	zu belegendes Muster

- lokale Variablen
  keine

- Rückgabewert

Name	Datentyp	Bedeutung
weiter	bool	Angabe, ob ein weiteres Zeichen überlesen werden muß

- Beschreibung
  `zuweisen()` ist eine Hilfsfunktion für `musteranalyse()`.Ein Bestandteil des Musters wird belegt. Zudem ist hier eine Optimierungsstrategie verwirklicht: Finden sich zwei aufeinanderfolgende Zeichenklassen mit Wiederholung, die dieselben „passenden" Zeichen haben, so werden sie zu einer Zeichenklasse zusammengefaßt. Das spart rekursive Funktionsaufrufe ein.

4.8.3 Funktion komplement()

- Prototyp

```
static void komplement(const char *c1, char *c2, const char *grund);
```

- Eingabeparameter

Name	Datentyp	Bedeutung
c1	const char*	Text, dessen Komplement gesucht ist.
grund	const char*	Grund-Zeichenmenge

- Ausgabeparameter

Name	Datentyp	Bedeutung
c2	char*	Komplement von c1 in grund

- lokale Variablen

Name	Datentyp	Bedeutung
i	int	Zählvariable
c	char	Zwischenspeicher
pos	int	Vergleichsposition

- Rückgabewert
  keiner
- Beschreibung
  `komplement()` ist eine Hilfsfunktion für `musteranalyse()`. Nach Ende dieser Funktion steht in `c2` das Komplement von `c1` bezüglich `grund`.

4.8.4 Funktion lese_zk()

- Prototyp

```
static int lese_zk(const char *suchtext, int *i,
                   const char *erlaubte_zeichen, char *dummy);
```

- Eingabeparameter

Name	Datentyp	Bedeutung
suchtext	const char*	Zeichenkette, die das Suchmuster enthält
erlaubte_zeichen	const char*	Zeichen des Alphabetes

- Ausgabeparameter

Name	Datentyp	Bedeutung
i	int*	aktuelle Position in `suchtext`
dummy	char*	einzulesende Zeichenklasse

- lokale Variablen

Name	Datentyp	Bedeutung
j	int	aktuelle Position in `dummy`
c	int	Zwischenspeicher

- Rückgabewert

Name	Datentyp	Bedeutung
keiner	int	Fehlercode

- Beschreibung
  `lese_zk()` ist auch eine Hilfsfunktion für `musteranalyse()`. Eine Zeichenklasse wird eingelesen und auf Gültigkeit überprüft. Sie ist nicht gültig, wenn sie nicht erlaubte Zeichen beinhaltet oder leer ist.

4.8.5 Funktion musteranalyse()

- Prototyp

```
static muster musteranalyse(const char *suchtext);
```

- Eingabeparameter

Name	Datentyp	Bedeutung
suchtext	const char*	Zeichenkette, die das Suchmuster enthält

- Ausgabeparameter
  keine

- lokale Variablen

Name	Datentyp	Bedeutung
erlaubte_zeichen	const char[]	Zeichen des Alphabetes
aktuelles_zeichen	char	aktuelles Zeichen in `suchtext`
dummy	char[]	Zwischenspeicher
dummy2	char[]	Zwischenspeicher
interpretation	bool	Angabe, ob Metazeichen interpretiert wird
weiter	bool	Angabe, ob ein weiteres Zeichen überlesen werden muß
i	int	Zählvariable
n	int	Länge von `suchtext`
rc	int	Fehlercode

- Rückgabewert

Name	**Datentyp**	**Bedeutung**
m	muster	ermitteltes Muster

- Beschreibung
  `musteranalyse` zerlegt den übergebenen Suchtext in Einzelbestandteile und speichert sie in der datenstruktur `muster` ab. Im Fehlerfalle ist die Angabe über die Anzahl der Bestandteile des Musters negativ.

- Nassi-Shneidermann-Diagramm

static muster **musteranalyse**(**const char** *suchtext)

Muster hat noch keine Bestandteile

solange Ende von suchstring und maximale Anzahl an Bestandteilen noch nicht erreicht

lies ein Zeichen

Interpretation

T

erlaubtes Zeichen bis auf Punkt

T

nur das aktu- elle Zeic- hen passt

Wie- derh- olung erm- itteln

ein Besa- tndte- il mehr

F

unzulaessiges Metazeichen

T

meld Sy- nt- ax- fe- hl- er

F

welches Zeichen eingelesen

','

ke- ine In- ter- pr- et- ati- on des na- ec- hs- ten Ze- ic- he- ns

'.'

alle erlau- bten Zeic- hen pass- en

Wie- derh- olung ermi- tteln

ein Best- andt- eil mehr

'['

kein Komple- ment

T

lies Zeich- enklas- se ein

Zeich- enklas- se wenn gueltig passt

Wiede- rholun- g ermi- tteln

ein Be- standt- eil mehr

F

lies Zeich- enklas- se ein

wenn gueltig passt Komp- lement

Wied- erholu- ng er- mitteln

ein Besta- ndteil mehr

de- fa- ult

F

erlaubtes Zeichen eingel- esen

T

nur dieses Zeich- en passt

Wied- erhol- ung er- mittel- n

ein Besta- ndteil mehr

das naech- ste Ze- ichen wird wieder inter- pretie- rt

F

mel- de Syn- taxf- ehler

4.8.6 Funktion suche_in_text()

- Prototyp

```
static void suche_in_text(muster m, const char *text, int position,
                          int zustand, bool *treffer, int *len);
```

- Eingabeparameter

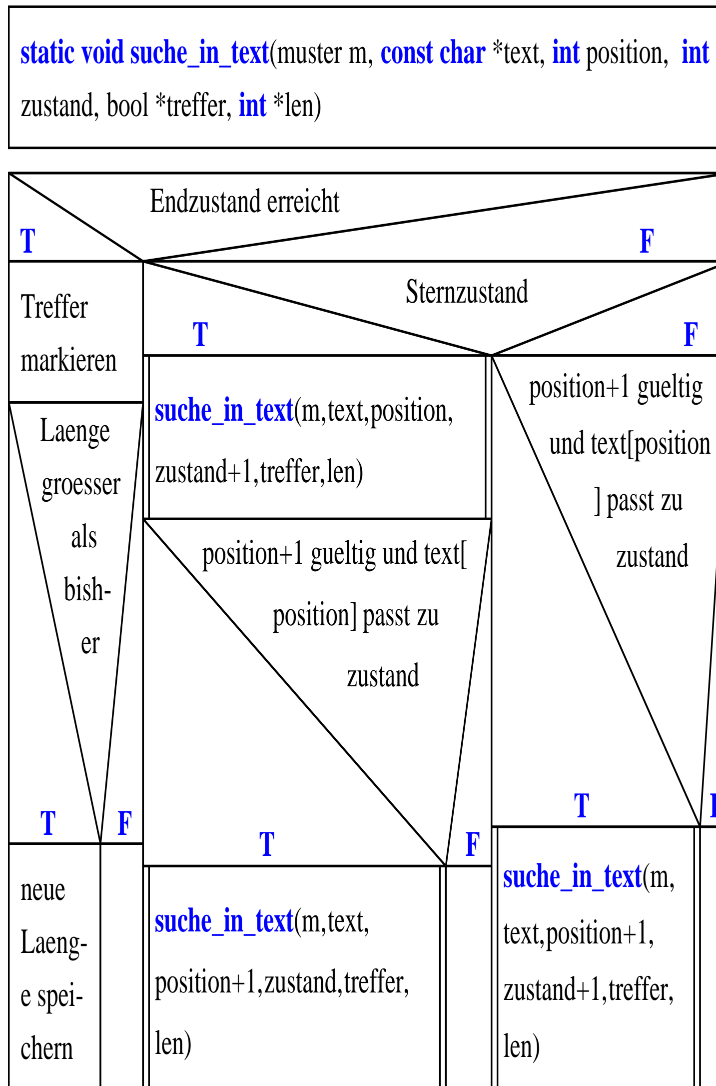
Name	Datentyp	Bedeutung
m	muster	zu suchendes Muster
text	const char*	zu durchsuchender Text
position	int	aktuelle Position in `text`
zustand	int	aktueller Zustand

- Ausgabeparameter

Name	Datentyp	Bedeutung
treffer	bool*	Angabe, ob das Muster gefunden wurde
len	int*	Endposition des gefundenen Textabschnittes

- lokale Variablen
  keine
- Rückgabewert
  keiner (Die Rückgabe der Ergebnisse erfolgt über die Ausgabeparameter.)
- Beschreibung
  `suche_in_text()` simuliert einen endlichen Automaten. Dessen Zustände entsprechen den Musterbestandteilen des zu suchenden Musters. An der übergebenen Position wird ermittelt, ob ein passender Textabschnitt vorliegt. Ist dies der Fall, so kann der Automat in den nächsten Zustand übergehen. Dies entspricht einem rekursiven Aufruf der Funktion `suche_in_text()` mit einem um 1 erhöhten Zustand. Bei „Sternzuständen“ wird zusätzlich mit dem gleichen Zustand und einer um 1 erhöhten Position im Text weitergesucht, da mehrere Zeichen hintereinander zu dem selben Zustand passen können. Kann der Automat auf diese Weise in den Endzustand gelangen (wird bis zu der Rekursionsstufe verzweigt, in der alle Musterbestandteile abgearbeitet sind), wird die Endposition dieses „Treffers“ festgehalten, sofern sie größer ist als die bisher größte, die ermittelt werden konnte.

- Nassi-Shneidermann-Diagramm



4.8.7 Hauptunterprogramm suche()

- Prototyp

```
loesung suche(const char *suchtext, const char *text);
```

- Eingabeparameter

Name	**Datentyp**	**Bedeutung**
suchtext	const char*	Beschreibung des Suchmusters
text	const char*	zu durchsuchender Text

- Ausgabeparameter
  keine

- lokale Variablen

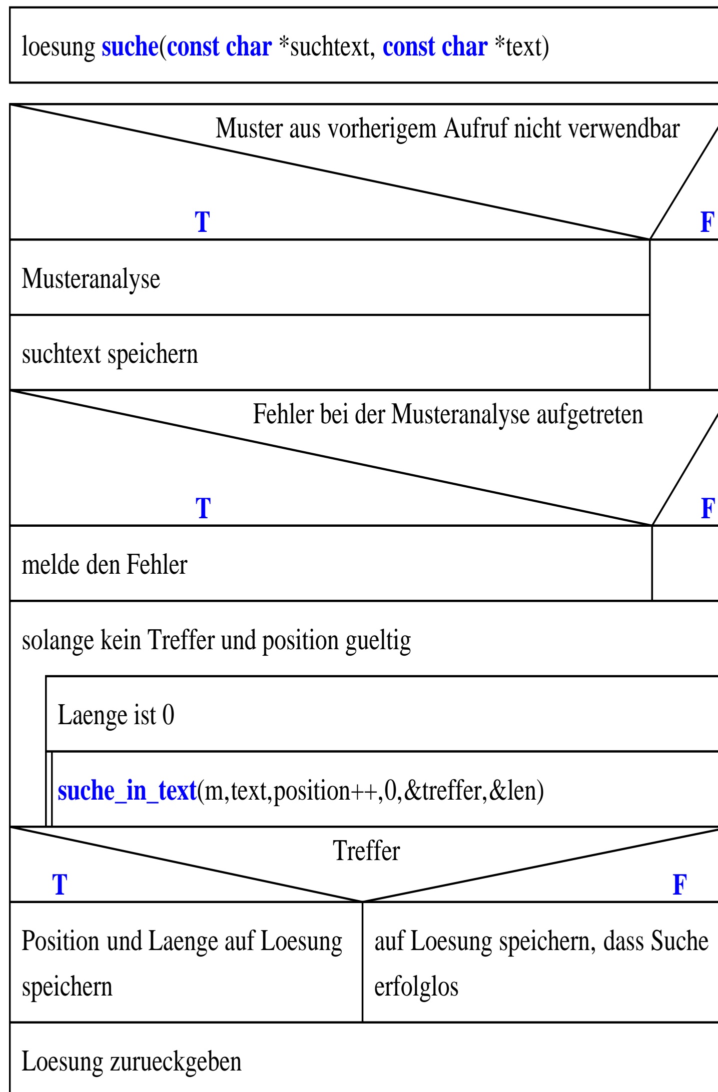
Name	**Datentyp**	**Bedeutung**
m	muster	Muster aus dem Voraufruf
vorheriger_suchtext	static char[]	Suchtext aus dem Voraufruf
position	int	aktuelle Suchposition
n	int	Länge von `text`
len	int	Länge des gefundenen Textabschnittes
treffer	bool	Angabe, ob das Muster gefunden wurde

- Rückgabewert

Name	**Datentyp**	**Bedeutung**
l	loesung	Ergebnisse der Textsuche

- Beschreibung
  Das Haupt-Untergrogramm `suche()` ruft `suche_im_text()` für jede Position im Text (am Anfang angefangen) auf, bis daß entweder ein Treffer gefunden wird oder das Ende des zu durchsuchenden Textes erreicht ist. Die Ergebnisse der Textsuche werden in der Datenstruktur `loesung` gespeichert. Im Fehlerfalle ist die Komponente von `l`, die die Anfangsposition beschreibt, negativ. Ansonsten werden die Anfangsposition und die Maximallänge des gefundenen Textabschnittes in `l` festgehalten.

- Nassi-Shneidermann-Diagramm



5 Benutzeranleitung

5.1 Entwicklungsumgebung

Rechnertyp	IBM RS/6000 7044 Model 270
Betriebssystem	AIX Version 4.3.3.0
Compiler	GNU-C-Compiler V2.95 for AIX

Compilationszeit	`real 0m1.06s`
	`user 0m0.35s`
	`sys 0m0.13s`

Ausführungszeit (für das Testbeispiel)	`real 0m0.04s`
	`user 0m0.00s`
	`sys 0m0.02s`

5.2 Compilation und Aufruf

5.2.1 Compilation

Das Programm wird mit dem Aufruf

```
gcc -o<programmname> main.c ein_ausgabe.c verarbeitung.c
```

übersetzt.

5.2.2 Aufruf

Der Aufruf zu dem Programm lautet:

```
<programmname> <eingabedatei> [ausgabedatei]
```

Die Eingabedatei muß zwingend spezifiziert werden. Sollte keine Ausgabedatei angegeben sein, so erfolgt die Ausgabe der Ergebnisse des Programmlaufs auf der Standardausgabe. Im Betriebssystem UNIX kann die Standardausgabe mittels der Shell in eine Datei umgelenkt werden. Dazu muß folgender Befehl verwendet werden:

```
<programmname> <eingabedatei> > <ausgabedatei>
```

Fehlermeldungen werden grundsätzlich auf der Standardfehlerausgabe ausgegeben. Auch hier ist eine Umlenkung über die Shell unter UNIX möglich, der Befehl lautet dann wie folgt:

```
<programmname> <eingabedatei> [ausgabedatei] 2><fehlerausgabedatei>
```

5.3 Fehlercodes

Makroname	Fehlernummer	Bedeutung
AUFRUF	-15	Falscher Programmaufruf
INDAT	-14	Öffnen der Eingabedatei fehlgeschlagen
OUTDAT	-13	Öffnen der Ausgabedatei fehlgeschlagen
SPEICHER	-12	Nicht genügend Speicher zum Allozieren von text
DAT_LEER	-11	Eingabedatei enthält keine Daten
ZUENDE	-10	Unerwartetes Ende der Eingabedatei
NO_COMMENT	-9	Kommentarzeile fehlt in der Eingabedatei
NICHT_GETRENNT	-8	Separator zwischen Muster und Text fehlt
CHAR_SUCHTEXT	-7	Ungültige Zeichen im Suchtext
CHAR_TEXT	-6	Ungültige Zeichen im Text
SYNTAX_SUCHSTR	-5	Syntax im Suchstring ist inkorrekt
FALSCH_IN_ZK	-4	Unerlaubte Zeichen in einer Zeichenklasse
DOPPELT_IN_ZK	-3	Durch doppelte Angabe von Zeichen in einer Zeichenklasse wird der Suchstring zu lang
LEERE_ZK	-2	Leere Zeichenklasse aufgetreten
NICHT_GEFUNDEN	-1	Muster kann nicht im Text gefunden werden

5.4 Regeln für die Eingabedatei

1. In jeder Eingabedatei steht genau ein Suchmuster und genau ein zu durchsuchender Text.
2. Sowohl der Text, der das Suchmuster beschreibt, als auch der zu durchsuchende Text werden mit genau einer Kommentarzeile eingeleitet.
3. Kommentarzeilen beginnen mit einem '#' und sind maximal 80 Zeichen lang.
4. Als terminierendes Symbol für das Suchmuster dient das Zeichen '$'.Es muß zwingend angegeben werden.
5. Das Suchmuster darf aus höchstens 20 Einzelbestandteilen zusammengesetzt sein. Sind es mehr als 20 Einzelbestandteile, so werden nur die ersten 20 in die Suche mit einbezogen, die restlichen Bestandteile bleiben unberücksichtigt.

Sollte mindstens eine dieser Regeln verletzt sein, kann das Beispiel nicht abgearbeitet werden.

6 Diskussion der Testbeispiele

6.1 Beispiele aus der Aufgabenstellung

6.1.1 Erstes Beispiel aus der Aufgabenstellung

```
#Erstes Beispiel aus der Aufgabenstellung:
[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*$
#Text:
Die Gravitationskonstante ist g = +0.667E-10 Meter hoch drei durch
Kilogramm mal Sekunde zum Quadrat.
```

Anhand dieses Beispiels möchte ich den genauen Ablauf meines Programms näher erläutern.

1. Eingabe:
   Zunächst wird eine Zeile aus der Eingabedatei komplett eingelesen. Da dies gelingt, wird danach überprüft, ob als erstes Zeichen der eingelesenen Zeile das Zeichen '`#`' eingelesen wurde. Dies ist der Fall, also werden solange Zeichen aus der Eingabe in die Zeichenkette `suchtext` gelesen, bis daß das Zeichen '`$`' eingelesen wird. Ein Abbruch hätte vorher stattgefunden, wenn ungültige Zeichen in der Eingabedatei vorgelegen hätten. Das Zeichen '`$`'hingegen kennzeichnet das „ordnungsgemäße" Ende des Textes, der das Suchmuster umschreibt. Zur weiteren Bearbeitung wird dann noch das Zeichen `'\0'` an `suchtext` angehangen, damit sich Standard-Stringfunktionen auf `suchtext` anwenden lassen.

   Danach wird eine weitere Zeile komplett aus der Eingabedatei eingelesen. Da auch dies gelingt, kann erneut überprüft werden, ob es sich hierbei um eine Kommentarzeile (führendes Zeichen '`#`') handelt. Auch diese Abfrage liefert „wahr". Nun ermittelt die Einleseprozedur die aktuelle Dateiposition und die Position des Dateiendes. Auf `anzahl` wird gespeichert, wieviele Zeichen es von der aktuellen Position bis zur Endposition in der Datei sind. Entsprechend groß muß der zu durchsuchende Text sein. Der Vektor `text` wird also dynamisch mit der Länge `anzahl` angelegt. Weil auch dies ohne Fehler geschehen kann, werden nun `(anzahl-1)` Zeichen gelesen (letztes Zeichen in der Datei kennzeichnet das Dateiende und wird nicht mit eingelesen) und auf Gültigkeit überprüft. Alle eingelesenen Zeichen sind gültig und können in den Vektor `text` übertragen werden.

   Nachdem aus den selben Gründen wie bei `suchtext` noch das Stringende-Zeichen an `text` angehängt worden ist, kann die Einleseroutine mit der Meldung, daß kein Fehler aufgetreten ist (Return-Code 0) verlassen werden.

2. Analyse des Suchmusters:
   Da die Eingabedaten korrekt sind, ruft `main()` das Hauptunterprogramm `suche()` mit den eingelesen Zeichenketten als Argumente auf. `suche()` stellt zunächst fest, daß der übergebene Suchtext noch nicht vorher bearbeitet worden ist (erster Aufruf) und ruft daraufhin die Funktion `musteranalyse()`

auf. Diese ermittelt die Einzelbestandteile des Suchmusters wie folgt:
Zuerst tritt das Zeichen '[' auf, also muß eine Zeichenklasse folgen. Diese wird mittels der Funktion `lese_zk()` eingelesen und danach mittels `zuweisen()` an den ersten Bestandteil des Musters zugewiesen. Ferner wird in `zuweisen()` erkannt, daß die Zeichenklasse keine Wiederholung aufweisen darf und folglich wird in die Komponente `wiederholung` der Wert `false` eingetragen. das nachfolgende Zeichen '0' wird als einfaches Zeichen ohne Wiederholung erkannt. Danach trifft `musteranalyse()` auf das Zeichen '\'. Dieses wird nicht weiter verarbeitet, sondern lediglich die logische Variable `interpretation` auf `false` gesetzt. Dadurch kann im nächsten Durchlauf der den Bearbeitungsteil von `musteranalyse()` umschließenden `while`-Schleife erkannt werden, daß das Zeichen '.' nicht als Metazeichen, sondern als erlaubtes Satzzeichen aufzufassen ist. Der Analyse-Algorithmus kann danach bis zum Ende von `suchtext` weiterlaufen, da keine Fehler auftreten. Am Ende von `musteranalyse()` stehen in der internen Datenstruktur `muster m` folgende Werte:

```
m.bestandteile = 9

m.teile[0].passende_zeichen = "+-"
m.teile[0].wiederholung = false

m.teile[1].passende_zeichen = "0"
m.teile[1].wiederholung = false

m.teile[2].passende_zeichen = "."
m.teile[2].wiederholung = false

m.teile[3].passende_zeichen = "123456789"
m.teile[3].wiederholung = false

m.teile[4].passende_zeichen = "0123456789"
m.teile[4].wiederholung = true

m.teile[5].passende_zeichen = "Ee"
m.teile[5].wiederholung = false

m.teile[6].passende_zeichen = "+-";
m.teile[6].wiederholung = false

m.teile[7].passende_zeichen = "0123456789";
m.teile[7].wiederholung = false

m.teile[8].passende_zeichen = "0123456789";
m.teile[8].wiederholung = true
```

Die Abarbeitung von `suchtext` hat keine Fehler ergeben, also kann `m` in der bestehenden Form an `suche()` zurückgegeben werden.

3. Suche im Text:
`suche()` ruft nach Beendigung von `musteranalyse()` `suche_in_text()` zunächst mit den Parametern `position 0` und `zustand 0` auf. Da der Zustand 0 kein „Sternzustand“ ist, wird überprüft, ob das Zeichen '`D`' zum Zustand 0 „paßt“. Dies ist nicht der Fall, so daß keine rekursiven Funktionsaufrufe erfolgen und `suche_in_text()` sofort wieder verlassen wird. Da kein „Treffer“ gefunden wurde, ruft `suche()` danach `suche_in_text` mit den Parametern `position 1` und `zustand 0` erneut auf. Wieder wird kein Treffer gefunden. Dies wiederholt sich in der `while`-Schleife von `suche()` so lange, bis `suche_in_text()` mit den Parametern `position 34` und `zustand 0` aufgerufen wird. Hier findet `suche_in_text()` ein Zeichen ('`+`'), welches zu Zustand 0 paßt. Da der Zustand 0 kein „Sternzustand“ ist, erfolgt lediglich ein rekursiver Aufruf von `suche_in_text()` mit den Parametern `position 35` und `zustand 1`. Auch hier wird ein passendes Zeichen ('`0`') gefunden, so daß in die nächste Rekursionsstufe (`position 36, zustand 2`) verzweigt werden kann. Ebenso verfährt der Algorithmus auch in den Zuständen 2 und 3, da '`.`' und '`6`' jeweils „passen“. In Zustand 4 erreicht `suche_in_text()` erstmalig einen „Sternzustand“. Dies hat zwei rekursive Funktionsaufrufe zur Folge: Erstens wird `suche_in_text()` mit einem um 1 erhöhten Zustand und der selben Textposition aufgerufen, um den Fall „nullmaliges Vorkommen der passenden Zeichen aus Zustand 4“ abzudecken. Zweitens erfolgt ein rekursiver Aufruf von `suche_in_text()` mit einer um 1 erhöhten Textposition und dem selben Zustand, damit der Fall „ein- oder mehrmaliges Vorkommen der passenden Zeichen aus Zustand 4“ abgearbeitet werden kann. Ein Zweig dieser rekursiven Funktionsaufrufe, nämlich der, der sofort einen neuen Zustand erzeugt hat, kommt in der nächsten Rekursionsstufe nicht weiter, da keines der Zeichen '`E`' oder '`e`' im Text steht. Der andere Aufruf arbeitet sich hingegen bis in Zustand 8 weiter vor, ohne daß zwischenzeitlich Abbrüche auf Grund nicht „passender“ Zeichen erfolgen würden. Er verzweigt sich sogar noch weiter, da Zustand 4 auch in der nächsten Rekursionsebene als „Sternzustand“ erkannt wird. In Zustand 8 ist dann wieder ein „Sternzustand“ erreicht. Hier „gabeln“sich die rekursiven Funktionsaufrufe wieder wie in in Zustand 4 beschrieben auf. Alle rekursiven Aufrufe von `suche_in_text()`, die in Zustand 8 getätigt werden, gelangen danach in Zustand 9. Diejenigen Aufrufe, die in Zustand 8 zunächst noch einmal mit Zustand 8 und einer um 1 erhöhten Textposition weiterarbeiten, liefern die maximale Länge, da auch das Zeichen '`0`', welches am Ende des erkannten Suchpatterns steht, erkannt wird. Damit ergibt sich die am weitesten vorgerückte Position zu 44. `suche()` ermittelt daraus die maximale Länge des gefundenen Textabschnittes zu $44 - 35 + 1 = 10$ und liefert sie zusammen mit der Anfangsposition 35 über die Datenstruktur `loesung l` an das Hauptprogramm zurück.

4. Ausgabe:
   `main()` stellt fest, daß `suche()` erfolgreich nach dem Muster im Text gesucht hat und reicht `loesung 1` zusammen mit den Eingabestrings an `ausgabe()` weiter. Es ergibt sich folgende Ausgabe:

```
Muster:

[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*

Text:

Die Gravitationskonstante ist g = +0.667E-10 Meter hoch drei durch
Kilogramm mal Sekunde zum Quadrat.


Treffer: Position 35, Laenge 10
```

Die Eingabedaten zu diesem Beispiel finden sich in der Datei `test_as1.inp`. Die Ausgabe steht in `test_as1.output`.

6.1.2 Weitere Beispiele aus der Aufgabenstellung

Die Eingabedaten zu den anderen vier in der Aufgabenstellung angegebenen Beispiele stehen in den Dateien `test_as2.inp`, `test_as3.inp`, `test_as4.inp` und `test_as5.inp`. Sie liefern jeweils keine Treffer. Die Ausgabedaten stehen in den Dateien `test_as2.output`, `test_as3.output`, `test_as4.output` und `test_as5.output`.

Eingabe des zweiten Beispiels aus der Aufgabenstellung:

```
#Zweites Beispiel aus der Aufgabenstellung:
[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*$
#Text:
Die Gravitationskonstante ist g = 0.667E-10 Meter hoch drei durch
Kilogramm mal Sekunde zum Quadrat.
```

Ausgabe des zweiten Beispiels aus der Aufgabenstellung:

```
Muster:

[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*

Text:

Die Gravitationskonstante ist g = 0.667E-10 Meter hoch drei durch
Kilogramm mal Sekunde zum Quadrat.


Kein Treffer!
```

Eingabe des dritten Beispiels aus der Aufgabenstellung:

```
#Drittes Beispiel aus der Aufgabenstellung:
[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*$
#Text:
0.5E-6
```

Ausgabe des dritten Beispiels aus der Aufgabenstellung:

```
Muster:

[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*

Text:

0.5E-6


Kein Treffer!
```

Eingabe des vierten Beispiels aus der Aufgabenstellung:

```
#Viertes Beispiel aus der Aufgabenstellung:
[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*$
#Text:
+1.234E+6
```

Ausgabe des vierten Beispiels aus der Aufgabenstellung:

```
Muster:

[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*

Text:

+1.234E+6


Kein Treffer!
```

Eingabe des fünften Beispiels aus der Aufgabenstellung:

```
#Fuenftes Beispiel aus der Aufgabenstellung:
[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*$
#Text:
+0.123e25
```

Ausgabe des fünften Beispiels aus der Aufgabenstellung:

```
Muster:

[+-]0\.[123456789][0123456789]*[Ee][+-][0123456789][0123456789]*

Text:

+0.123e25


Kein Treffer!
```

6.2 Weitere Normal- und Extrembeispiele

6.2.1 Beispiel mit Satzzeichen und Umlauten

Das nächste Beispiel zeigt ein Normalbeispiel. Das besondere an diesem Beispiel ist, daß hier der Zeichenvorrat an Satz- und Sonderzeichen aus dem Alphabet voll ausgeschöpft wird. Ferner zeigt das Beispiel den Umgang des Programmes mit dem Komplement einer Zeichenklasse. Die Eingabedatei lautet `test_nb1.inp` und die Ausgabedatei `test_nb1.output`.

Eingabe des Normalbeispiels:

```
#Normalbeispiel, Sonderzeichen kommen vor:
www\.[^ÄÖÜäöü!?.,:;]*\.de$
#Text:
Suche nach einer deutschen Website, wie etwa www.schachverein-wk.de.
```

Ausgabe des Normalbeispiels:

```
Muster:

www\.[^ÄÖÜäöü!?.,:;]*\.de

Text:

Suche nach einer deutschen Website, wie etwa www.schachverein-wk.de.


Treffer: Position 46, Laenge 22
```

6.2.2 Maximalbeispiel bezüglich der Anzahl an Musterbestandteilen

Dieses Beispiel zeichnet sich dadurch aus, daß getestet wird, ob das Programm richtig mit der maximalen Anzahl möglicher Musterbestandteile umgeht. Es sind 20 Musterbestandteile zu suchen. Die Eingabe steht in `test_eb1.inp`, die Ausgabe in `test_eb1.output`.

Eingabe des Maximalbeispiels:

```
#Maximalbeispiel bezueglich Anzahl der Musterbestandteile:
[0123][0123456789]\.[01][0123456789]\.[12][0123456789][0123456789][0123456789][ ,][012][0123456
789]:[012345][0123456789]:[012345][0123456789]h$
#Text:
Suche nach einem Datum in der deutschen Form: tt.mm.jjjj !
Dec-01-2000 (englische Form)
01.12.2000 16:00:00h (deutsche Form)
```

Ausgabe des Maximalbeispiels:

```
Muster:

[0123][0123456789]\.[01][0123456789]\.[12][0123456789][0123456789][0123456789][ ,][012][0123456
789]:[012345][0123456789]:[012345][0123456789]h

Text:
```

```
Suche nach einem Datum in der deutschen Form: tt.mm.jjjj !
Dec-01-2000 (englische Form)
01.12.2000 16:00:00h (deutsche Form)


Treffer: Position 89, Laenge 20
```

6.2.3 Extrembeispiel bezüglich der Länge des zu durchsuchenden Textes:

Im folgenden Beispiel wird ein sehr langer Text durchsucht. Das Beispiel ist aber noch aus einem anderen Grund interessant: Hier zeigt sich, daß es wichtig ist, in jedem Fall (nicht nur im dem Falle, daß obere Rekursionsebenen keinen Treffer liefern) einen rekursiven Aufruf in einem „Sternzustand“ zu tätigen. Würde man dies hier nicht berücksichtigen, so fände der Algorithmus ein zu kurzes Textstück (nur 775 Zeichen lang). Die zugehörige Eingabedatei ist `test_eb2.inp`, die Ausgabedatei `test_eb2.output`.

Eingabe des Extrembeispiels:

```
#Extrembeispiel bezueglich Laenge des zu durchsuchenden Textes:
[äöü].*[!?:;.,] die [^;]*$
#Franz Kafka, 1919:
Auf der Galerie

Wenn irgendeine hinfällige, lungensüchtige Kunstreiterin in der
Manege auf schwankendem Pferd vor einem unermüdlichen Publikum vom
peitschenschwingenden erbarmungslosen Chef monatelang ohne
Unterbrechung im Kreise rundum getrieben würde, auf dem Pferde
schwirrend, Küsse werfend, in der Taille sich wiegend, und wenn dieses
Spiel unter dem nichtaussetzenden Brausen des Orchesters und der
Ventilatoren in die immerfort weiter sich öffnende graue Zukunft sich
fortsetzte, begleitet vom vergehenden und neu anschwellenden
Beifallsklatschen der Hände, die eigentlich Dampfhämmer sind -
vielleicht eilte dann ein junger Galeriebesucher die lange Treppe
durch alle Ränge hinab, stürzte in die Manege, riefe das: Halt! durch
die Fanfaren des immer sich anpassenden Orchesters.

Da es aber nicht so ist; eine schöne Dame, weiss und rot,
hereinfliegt, zwischen den Vorhängen, welche die stolzen Livrierten
vor ihr öffnen; der Direktor, hingebungsvoll ihre Augen suchend, in
Tierhaltung ihr entgegenatmet; vorsorglich sie auf den Apfelschimmel
hebt, als wäre sie seine über alles geliebte Enkelin, die sich auf
gefährliche Fahrt begibt; sich nicht entschliessen kann, das
Peitschenzeichen zu geben; schliesslich in Selbstüberwindung es
knallend gibt; neben dem Pferde mit offenem Munde einherläuft; die
Sprünge der Reiterin scharfen Blickes verfolgt; ihre Kunstfertigkeit
kaum begreifen kann; mit englischen Ausrufen zu warnen versucht; die
reifenhaltenden Reitknechte wütend zu peinlichster Achtsamkeit
ermahnt; vor dem grossen Saltomortale das Orchester mit aufgehobenen
Händen beschwört, es möge schweigen; schliesslich die Kleine vom
zitternden Pferde hebt, auf beide Backen küsst und keine Huldigung des
Publikums für genügend erachtet; während sie selbst, von ihm gestützt,
hoch auf den Fussspitzen, vom Staub umweht, mit ausgebreiteten Armen,
zurückgelehntem Köpfchen ihr Glück mit dem ganzen Zirkus teilen will -
```

```
da dies so ist, legt der Galeriebesucher das Gesicht auf die Brüstung
und, im Schlussmarsch wie in einen schweren Traum versinkend, weint
er, ohne es zu wissen.
```

Ausgabe des Extrembeispiels:

```
Muster:

[äöü].*[!?:;.,] die [^;]*

Text:

Auf der Galerie

Wenn irgendeine hinfällige, lungensüchtige Kunstreiterin in der
Manege auf schwankendem Pferd vor einem unermüdlichen Publikum vom
peitschenschwingenden erbarmungslosen Chef monatelang ohne
Unterbrechung im Kreise rundum getrieben würde, auf dem Pferde
schwirrend, Küsse werfend, in der Taille sich wiegend, und wenn dieses
Spiel unter dem nichtaussetzenden Brausen des Orchesters und der
Ventilatoren in die immerfort weiter sich öffnende graue Zukunft sich
fortsetzte, begleitet vom vergehenden und neu anschwellenden
Beifallsklatschen der Hände, die eigentlich Dampfhämmer sind -
vielleicht eilte dann ein junger Galeriebesucher die lange Treppe
durch alle Ränge hinab, stürzte in die Manege, riefe das: Halt! durch
die Fanfaren des immer sich anpassenden Orchesters.

Da es aber nicht so ist; eine schöne Dame, weiss und rot,
hereinfliegt, zwischen den Vorhängen, welche die stolzen Livrierten
vor ihr öffnen; der Direktor, hingebungsvoll ihre Augen suchend, in
Tierhaltung ihr entgegenatmet; vorsorglich sie auf den Apfelschimmel
hebt, als wäre sie seine über alles geliebte Enkelin, die sich auf
gefährliche Fahrt begibt; sich nicht entschliessen kann, das
Peitschenzeichen zu geben; schliesslich in Selbstüberwindung es
knallend gibt; neben dem Pferde mit offenem Munde einherläuft; die
Sprünge der Reiterin scharfen Blickes verfolgt; ihre Kunstfertigkeit
kaum begreifen kann; mit englischen Ausrufen zu warnen versucht; die
reifenhaltenden Reitknechte wütend zu peinlichster Achtsamkeit
ermahnt; vor dem grossen Saltomortale das Orchester mit aufgehobenen
Händen beschwört, es möge schweigen; schliesslich die Kleine vom
zitternden Pferde hebt, auf beide Backen küsst und keine Huldigung des
Publikums für genügend erachtet; während sie selbst, von ihm gestützt,
hoch auf den Fussspitzen, vom Staub umweht, mit ausgebreiteten Armen,
zurückgelehntem Köpfchen ihr Glück mit dem ganzen Zirkus teilen will -
da dies so ist, legt der Galeriebesucher das Gesicht auf die Brüstung
und, im Schlussmarsch wie in einen schweren Traum versinkend, weint
er, ohne es zu wissen.


Treffer: Position 38, Laenge 1106
```

6.3 Sonderbeispiele

6.3.1 Sonderbeispiel mit mehr als 20 Musterbestandteilen

Im folgenden Beispiel weist das Suchmuster mehr als 20 Bestandteile auf. Vom Programm werden in diesem Fall nur die ersten 20 Bestandteile verarbeitet.

Die Eingabe dieses Beispiels steht in der Datei `test_sb1.inp`, die Ausgabe in `test_sb1.output`.

Eingabe des ersten Sonderbeispiels:

```
#Mehr als 20 Suchausdruecke, nur die ersten 20 werden bearbeitet:
1234567890123456789012345$
#Text:
123456789012345678901234567890
```

Ausgabe des ersten Sonderbeispiels:

```
Muster:

1234567890123456789012345

Text:

123456789012345678901234567890

Treffer: Position 1, Laenge 20
```

6.3.2 Sonderbeispiel mit mehreren gleichen Zeichenklassen mit Wiederholung nacheinander

Dieses Beispiel verdeutlicht die Effizienz der Optimierungsstrategie. Es wird nach einem Muster gesucht, in dem zehn aufeinanderfolgende Zeichenketten mit Wiederholung gleichen Inhalts vorkommen. Würde man die Optimierungsstrategie, mehrere aufeinanderfolgende Zeichenklassen mit Wiederholung gleichen Inhalts zusammenzufassen nicht verfolgen, so ergäbe sich bei diesem Beispiel eine übermäßig hohe Zahl an rekursiven Funktionsaufrufen, die die Laufzeit drastisch in die Höhe treiben würde. So jedoch kann das Beispiel in sehr kurzer Zeit abgearbeitet werden. Die zugehörige Eingabedatei lautet: `test_sb2.inp`, die Ausgabedatei `test_sb2.output`.

Eingabe des zweiten Sonderbeispiels:

```
#Sonderbeispiel, mehrere Zeichenklassen mit Wiederholung und selbem Inhalt:
[123456789][0123456789]*\.[0123456789]*[0123456789]*[0123456789]*[0123456789]*[0123456789]*[012
3456789]*[0123456789]*[0123456789]*[0123456789]*[0123456789]*$
#Suche nach einer Dezimalzahl mit keiner, einer oder mehrerer Nachkommastellen:
15.
```

Ausgabe des zweiten Sonderbeispiels:

```
Muster:

[123456789][0123456789]*\.[0123456789]*[0123456789]*[0123456789]*[0123456789]*[0123456789]*[012
3456789]*[0123456789]*[0123456789]*[0123456789]*[0123456789]*

Text:

15.


Treffer: Position 1, Laenge 3
```

6.3.3 Sonderbeispiel: keinmaliges Vorkommen eines Zeichens

Hier zeigt sich, daß der rekursive Aufruf mit einem Zustand weiter im Falle eines „Sternzustands“ auf jeden Fall mit der selben Position als Argument erfolgen muß, damit auch ein keinmaliges Vorkommen eines Musterbestandteil erkannt wird. Die Eingabe steht in `test_sb3.inp`, die Ausgabe in `test_sb3.output`.

Eingabe des dritten Sonderbeispiels:

```
#Sonderbeispiel, keinmaliges Vorkommen eines Zeichens:
Meteu*r$
#Text:
Die Gravitationskonstante ist g = +0.667E-10 Meter hoch drei durch Kilogramm mal Sekunde
zum Quadrat.
```

Ausgabe des dritten Sonderbeispiels:

```
Muster:

Meteu*r

Text:

Die Gravitationskonstante ist g = +0.667E-10 Meter hoch drei durch Kilogramm mal Sekunde
zum Quadrat.


Treffer: Position 46, Laenge 5
```

6.3.4 Sonderbeispiel mit leerem Text

Als letztes Sonderbeispiel habe ich ein Beispiel mit einem leeren Text gewählt, der zu durchsuchen ist. Natürlich kann hier nichts gefunden werden, aber der Test war hauptsächlich dazu da, um zu überprüfen, ob in einem solchen Falle die dynamische Speicherallokation funktioniert. Eingabedatei: `test_sb4.inp`, Ausgabedatei: `test_sb4.output`.

Eingabe des vierten Sonderbeispiels:

```
#Muster:
a$
#Text:
```

Ausgabe des vierten Sonderbeispiels:

```
Muster:

a

Text:



Kein Treffer!
```

6.4 Fehlerbeispiele

Bei dieser Aufgabenstellung gibt es eine Menge möglicher Fehlerquellen; exemplarisch habe ich zwei ausgewählt.

6.4.1 Fehlerbeispiel mit nicht erlaubtem Zeichen

Das erste Fehlerbeispiel behandelt einen syntaktischen Fehler. Es wird ein Zeichen ('*') durch den '\' maskiert, das aber trotz Maskierung ungültig ist, da es nicht in der Menge der erlaubten Zeichen vorkommt.
Generell ist zu den Fehlerbeispielen anzumerken, daß die Fehlermeldungen normalerweise auf der Standard-Fehlerausgabe erscheinen, ich diese jedoch jeweils in Files umgelenkt habe. Die Eingabedatei zu diesem Beispiel lautet: `test_fb1.inp`, die Ausgabedatei `test_fb1.output`.

Eingabe des ersten Fehlerbeispiels:

```
#Fehlerbeispiel: Maskiertes Zeichen trotzdem ungueltig
[0123456789] \* [0123456789]$
#Ist eine Multiplikation dabei ?
3 - 5
4 + 7
```

Ausgabe des ersten Fehlerbeispiels:

```
Syntax des Suchstrings nicht korrekt !
```

6.4.2 Fehlerbeispiel mit leerer Zeichenklasse

Das zweite Fehlerbeispiel zeigt einen logischen (semantischen) Fehler. Es tritt eine leere Zeichenklasse auf, was nach Aufgabenstellung nicht erlaubt ist. Die Eingabe steht in `test_fb2.inp`, die Ausgabe in `test_fb2.output`.

Eingabe des zweiten Fehlerbeispiels:

```
#Fehlerbeispiel mit leerer Zeichenklasse:
[123][^]$
#Text:
19.8
```

Ausgabe des zweiten Fehlerbeispiels:

```
Leere Zeichenklasse aufgetreten !
```

A Änderungen zum Vorentwurf vom ersten Tag

A.1 Abweichungen vom Entwurf am ersten Tag

- Die Länge des zu durchsuchenden Textes wird vom Programm selbst ermittelt. Damit fällt das zuerst vorgesehene zweite Argument (Länge des zu durchsuchenden Textes) weg.
- Der Vektor mit den erlaubten Zeichen wird nicht wie vorgesehen zentral in einer Header-Datei definiert, sondern lokal in den Funktionen `eingabe()` und `musteranalyse()`.

A.2 Korrekturen zum Entwurf am ersten Tag

- Um die maximale Länge des passenden Textabschnittes herauszufinden, darf der rekursive Funktionsaufruf im „Sternzustand“ mit dem gleichen Zustand und einem Zeichen weiter im Text nicht nur dann erfolgen, wenn bis zum Rücksprung an diese Stelle kein Treffer gefunden wird, sondern muß stets erfolgen.
- Wenn von einem „Sternzustand“ zu einem anderen Zustand verzweigt wird, darf die Position im Text nicht erhöht werden, da auch nullmaliges Vorkommen eines passenden Zeichens zulässig ist.

A.3 Ergänzungen zum Entwurf am ersten Tag

- Die Berechnung der maximalen Länge des passenden Textabschnitts wurde in meinem Entwurf am ersten Tag vergessen.
- An die Funktion `ausgabe()` müssen zusätzlich zu den beiden spezifizierten Parametern noch die beiden eingelesenen Strings übergeben werden, da diese auch ausgegeben werden sollen (siehe Ausgabebeispiel in der Aufgabenstellung). Die Schnittstelle ist entsprechend zu erweitern.

B Hilfsmittel

Zur Bearbeitung der praktischen Arbeit habe ich folgende Hilfsmittel benutzt:

- LaTeX - eine Einführung und ein bißchen mehr
  Manuela Jürgens
  FZJ-ZAM-BHB-0134

- LaTeX - Fortgeschrittene Anwendungen
  oder: Neues von den Hobbits...
  Manuela Jürgens
  FZJ-ZAM-BHB-0135

- Programmierung in C
  Vorlesungsskript
  Günter Egerer
  FZJ-ZAM-BHB-0140

C Erklärung

Die praktische Arbeit habe ich selbständig erstellt und keine Hilfe in Anspruch genommen bei:

a. Konzepterstellung und Programmierung

b. inhaltliche Gestaltung der Dokumentation

c. Auswahl und Diskussion der Testbeispiele.

Die als Arbeitshilfe benutzte Literatur ist in der Arbeit oder in einem Anhang vollständig aufgeführt.

Datum: 01. Dezember 2000

D Source-Code ANSI-C

D.1 Header-Datei global_header.h

```
#ifndef _GLOBAL_H_INCLUDED_  /* Schutz vor mehrmaligem Einfuegen */
  /***************************************************************************/
  /* Header-Datei global_header.h                                            */
  /***************************************************************************/
  /*                                                                         */
  /* Aufgabe:                                                                */
  /* --------                                                                */
  /* Globale Datenstrukturen zentral vereinbaren, noetige System-Header-    */
  /* dateien einbinden, globale Makros setzen                                */
  /*                                                                         */   10
  /* Enthaltene Datenstrukturen:                                             */
  /* ---------------------------                                             */
  /* bool                        Modellierung eines logischen Datentyps      */
  /* musterbestandteil           Einzelner regulaerer Ausdruck, aus dem      */
  /*                             Muster zusammengesetzt werden koennen       */
  /* muster                      Gesamtmuster, welches sich aus Bestand-     */
  /*                             teilen zusammensetzt                        */
  /* loesung                     Datenstruktur fuer die Speicherung der      */
  /*                             zwei Bestandteile einer Loesung             */
  /*                                                                         */   20
  /* Enthaltene Makrodefinitionen:                                           */
  /* -----------------------------                                           */
  /* ALPHA                       Anzahl erlaubter Zeichen im Text            */
  /* MAXBESTAND                  maximale Anzahl von Musterbestandteilen     */
  /***************************************************************************/

 #define _GLOBAL_H_INCLUDED_

 #include <stdio.h>
 #include <stdlib.h>                                                             30
 #include <string.h>

 #define ALPHA 81
 #define MAXBESTAND 20

 typedef enum{false, true} bool;

 typedef struct
         {
          char passende_zeichen[ALPHA+1];                                        40
          bool wiederholung;
         } musterbestandteil;

 typedef struct
         {
          int bestandteile;
          musterbestandteil teile[MAXBESTAND];
         } muster;

 typedef struct                                                                  50
         {
          int laenge;
          int anfang;
```

```
        } loesung;

#endif /* _GLOBAL_H_INCLUDED_ */
```

D.2 Header-Datei myerrors.h

```
#ifndef _MYERRORS_H_INCLUDED_
  /*************************************************************************/
  /* Header-Datei myerrors.h                                               */
  /*************************************************************************/
  /*                                                                       */
  /* Aufgabe:                                                              */
  /* --------                                                              */
  /* Makros fuer die bessere Lesbarkeit der auftretenden Fehlercodes      */
  /* bereitstellen.                                                        */
  /*                                                                       */   10
  /* Enthaltene Prototypen:                                                */
  /* ----------------------                                                */
  /* keine                                                                 */
  /*                                                                       */
  /* Enthaltene Makrodefinitionen:                                         */
  /* -----------------------------                                         */
  /*                                                                       */
  /* AUFRUF                    Falscher Programmaufruf                     */
  /* INDAT                     Oeffnen der Eingabedatei fehlgeschlagen     */
  /* OUTDAT                    Oeffnen der Ausgabedatei fehlgeschlagen     */   20
  /* SPEICHER                  Nicht genuegend Speicher zum Allozieren     */
  /*                           von text                                    */
  /* DAT_LEER                  Eingabedatei enthaelt keine Daten           */
  /* ZUENDE                    Unerwartetes Ende der Eingabedatei          */
  /* NO_COMMENT                Kommentarzeile fehlt in der Eingabedatei    */
  /* NICHT_GETRENNT            Separator zwischen Muster und text fehlt    */
  /* CHAR_SUCHTEXT             Ungueltige Zeichen im Suchtext              */
  /* CHAR_TEXT                 Ungueltige Zeichen im Text                  */
  /* SYNTAX_SUCHSTR            Syntax im Suchstring ist inkorrekt          */
  /* FALSCH_IN_ZK              Unerlaubte Zeichen in einer Zeichenklasse   */   30
  /* DOPPELT_IN_ZK             Durch doppelte Angabe von Zeichen in        */
  /*                           einer Zeichenklasse wird der Suchstring     */
  /*                           zu lang                                     */
  /* LEERE_ZK                  Leere Zeichenklasse aufgetreten             */
  /* NICHT_GEFUNDEN            Muster kann nicht im Text gefunden werden   */
  /*************************************************************************/

  #define _MYERRORS_H_INCLUDED_

  #define AUFRUF (-15)                                                           40
  #define INDAT (-14)
  #define OUTDAT (-13)
  #define SPEICHER (-12)
  #define DAT_LEER (-11)
  #define ZUENDE (-10)
  #define NO_COMMENT (-9)
  #define NICHT_GETRENNT (-8)
  #define CHAR_SUCHTEXT (-7)
  #define CHAR_TEXT (-6)
  #define SYNTAX_SUCHSTR (-5)                                                    50
  #define FALSCH_IN_ZK (-4)
  #define DOPPELT_IN_ZK (-3)
  #define LEERE_ZK (-2)
  #define NICHT_GEFUNDEN (-1)
```

```
#endif /* _MYERRORS_H_INCLUDED_ */
```

D.3 Modul main.c

```
/**************************************************************************/
/*                                                                        */
/*                  Praktische Arbeit zur Pruefung zum                    */
/*                                                                        */
/*                 MATHEMATISCH-TECHNISCHEN ASSISTENTEN                   */
/*                                                                        */
/**************************************************************************/
/*                                                                        */
/*               Name                : Thorsten Dickhaus                 */
/*               IHK-Kennziffer      : 113                               */   10
/*               Programmiersprache  : ANSI-C                            */
/*               Compiler            : GNU-C-Compiler V2.95 for AIX      */
/*               Rechner             : IBM RS/6000 7044 Model 270        */
/*               Betriebssystem      : AIX 4.3.3.0                       */
/*               Datum               : 27.11.2000 bis 01.12.2000         */
/*                                                                        */
/**************************************************************************/
/**************************************************************************/
/* Modul main.c                                                           */
/**************************************************************************/   20
/*                                                                        */
/* Aufgabe:                                                               */
/* --------                                                               */
/* Dieses Modul enthaelt das Hauptprogramm main(), welches den Programm-  */
/* ablauf steuert.                                                        */
/*                                                                        */
/* Enthaltene Funktionen:                                                 */
/* ----------------------                                                 */
/* void fehlerbehandlung(int fehler, FILE *in, int zaehler, FILE *out,    */
/*                       char *text)                                      */   30
/* int main(int zaehler, char **argumente)                                */
/**************************************************************************/


#include "global_header.h"
#include "myerrors.h"
#include "ein_ausgabe.h"
#include "verarbeitung.h"

                                                                               40

/**************************************************************************/
/* Funktion "fehlerbehandlung"                                            */
/**************************************************************************/
/*                                                                        */
/* Zweck:                                                                 */
/* ------                                                                 */
/* Angeforderte Ressourcen im Fehlerfall freigeben und den Benutzer ueber */
/* den Fehler informieren                                                 */
/*                                                                        */   50
/* Eingabeparameter:                                                      */
/* -----------------                                                      */
/* int fehler           Nummer (Art) des aufgetretenen Fehlers            */
/* FILE *in             Zeiger auf die zu schiessende Eingabedatei        */
/* int zaehler          Anzahl der an das Programm uebergebenen Argumente */
```

```
/* FILE *out             Zeiger auf die Ausgabedatei oder auf die Stan-    */
/*                       dardausgabe (benutzerdefiniert)                   */
/* char *text            Zeiger auf den dynamisch angelegten Speicherplatz */
/*                       zur Speicherung des zu durchsuchenden Textes      */
/*                                                                         */   60
/* Ausgabeparameter:                                                       */
/* -----------------                                                       */
/* keine                                                                   */
/*                                                                         */
/* Rueckgabewert:                                                          */
/* --------------                                                          */
/* keiner                                                                  */
/***************************************************************************/

void fehlerbehandlung(int fehler, FILE *in, int zaehler, FILE *out,            70
                      char *text)
{
 fehlerausgabe(fehler);
 fclose(in);
 if(zaehler == 3)
   fclose(out);
 if(text != NULL)
   free(text);

 return;                                                                       80
}



/***************************************************************************/
/* Funktion "main"                                                         */
/***************************************************************************/
/*                                                                         */
/* Zweck:                                                                  */
/* ------                                                                  */   90
/* Hauptprogramm. Steuert den Programmablauf und die Ein-/Ausgabe.         */
/*                                                                         */
/* Eingabeparameter:                                                       */
/* -----------------                                                       */
/* int zaehler           Anzahl der an das Programm uebergebenen Parameter */
/* char **argumente      Zeichenketten-Vektor, der die an das Programm     */
/*                       uebergebenen Parameter enthaelt                   */
/* Ausgabeparameter:                                                       */
/* -----------------                                                       */
/* keine                                                                   */   100
/*                                                                         */
/* Rueckgabewert:                                                          */
/* --------------                                                          */
/* vom Typ int           Returncode fuer die aufrufende Umgebung           */
/***************************************************************************/
int main(int zaehler, char **argumente)
{
 char suchtext[(ALPHA+4)*MAXBESTAND+1];
 char *text = NULL;
 FILE *in, *out;                                                               110
 int fc;
 loesung l;
```

```
 if((zaehler != 2) && (zaehler != 3))            /* falscher Programmaufruf */
   {
    fehlerausgabe(AUFRUF);
    exit(EXIT_FAILURE);
   }

 in = fopen(argumente[1], "r");                                                   120
 if(!in)                               /* Eingabedatei nicht zu oeffnen */
   {
    fehlerausgabe(INDAT);
    exit(EXIT_FAILURE);
   }

 if(zaehler == 3)
   out = fopen(argumente[2], "w");
 else
   out = stdout;                                                                  130

 if(!out)                              /* Ausgabedatei nicht zu oeffnen */
   {
    fehlerausgabe(OUTDAT);
    exit(EXIT_SUCCESS);
   }

 fc = eingabe(suchtext, &text, in);
 if(fc)
   {                                                                              140
    fehlerbehandlung(fc, in, zaehler, out, text);
    exit(EXIT_FAILURE);
   }

 l = suche(suchtext, text);
 if(l.anfang < -1)
   {
    fehlerbehandlung(l.anfang, in, zaehler, out, text);
    exit(EXIT_FAILURE);
   }                                                                              150
 else
   ausgabe(suchtext, text, l, out);

 fclose(in);
 if(zaehler == 3)
   fclose(out);
 if(text != NULL)
   free(text);
 exit(EXIT_SUCCESS);
}                                                                                 160
```

D.4 Header-Datei ein_ausgabe.h

```
#ifndef _I_O_H_INCLUDED_
  /*************************************************************************/
  /* Header-Datei  ein_ausgabe.h                                           */
  /*************************************************************************/
  /*                                                                       */
  /* Aufgabe:                                                              */
  /* --------                                                              */
  /* Dem Hauptprogramm die Schnittstellen zu den benoetigten Ein-/Ausgabe-  */
  /* funktionen zur Verfuegung stellen                                     */
  /*                                                                       */     10
  /* Enthaltene Prototypen:                                                */
  /* ----------------------                                                */
  /* extern int eingabe(char *suchtext, char **text, FILE *in);            */
  /* extern int ausgabe(const char *suchtext, const char *text, loesung l,  */
  /*                    FILE *out);                                        */
  /* extern void fehlerausgabe(int fehler);                                */
  /*                                                                       */
  /* Enthaltene Makrodefinitionen:                                         */
  /* -----------------------------                                         */
  /* keine                                                                 */     20
  /*************************************************************************/


  #define  _I_O_H_INCLUDED_
  #include "global_header.h"

  extern int eingabe(char *suchtext, char **text, FILE *in);
  extern int ausgabe(const char *suchtext, const char *text, loesung l,
                     FILE *out);
  extern void fehlerausgabe(int fehler);                                          30

#endif /* _I_O_H_INCLUDED_ */
```

D.5 Modul ein_ausgabe.c

```
/***************************************************************************/
/* Modul ein_ausgabe.c                                                     */
/***************************************************************************/
/*                                                                         */
/* Aufgabe:                                                                */
/* --------                                                                */
/* Dieses Modul stellt die benoetigten Ein-/Ausgabe-Routinen zur           */
/* Verfuegung.                                                             */
/*                                                                         */
/* Enthaltene Funktionen:                                                  */
/* ----------------------                                                  */
/* int eingabe(char *suchtext, char **text, FILE *in)                      */
/* int ausgabe(const char *suchtext, const char *text, loesung l,          */
/*             FILE *out)                                                  */
/* void fehlerausgabe(int fehler)                                          */
/***************************************************************************/


#include "global_header.h"
#include "myerrors.h"



/***************************************************************************/
/* Funktion "eingabe"                                                      */
/***************************************************************************/
/*                                                                         */
/* Zweck:                                                                  */
/* ------                                                                  */
/* Einlesen des Suchtextes und des zu durchsuchenden Textes                */
/*                                                                         */
/* Eingabeparameter:                                                       */
/* -----------------                                                       */
/* FILE *in               Zeiger auf die Eingabedatei                      */
/*                                                                         */
/* Ausgabeparameter:                                                       */
/* -----------------                                                       */
/* char *suchtext         Zeichenkette, auf die der Suchtext eingelesen    */
/*                        werden soll                                      */
/* char **text            Zeiger auf die Zeichenkette, auf die der zu      */
/*                        durchsuchende Text eingelesen werden soll        */
/*                                                                         */
/* Rueckgabewert:                                                          */
/* --------------                                                          */
/* int rc                 Fehlercode                                       */
/***************************************************************************/
int eingabe(char *suchtext, char **text, FILE *in)
{
 const char erlaubte_zeichen[ALPHA+1] =
 "ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜabcdefghijklmnopqrstuvwxyzäöü"
 "0123456789 ()+-=\n.,!?:;";
 const char metazeichen[6+1] = ".*[]^\\";

 char dummy[80];
 char separator = '$';
```

```
int c,i, rc;
int anzahl;
long aktuelle_position, endposition;

rc = fscanf(in, "%80[^\n]", dummy);                                              60
fgetc(in);
if(rc != 1)
  return(DAT_LEER);       /* keine verwertbaren Daten in der Eingabedatei */

if(dummy[0] != '#')
  return(NO_COMMENT);     /* Kommentarzeile fehlt                         */

i = 0;
while(i < ((ALPHA+4)*MAXBESTAND) && (c  != separator))
     {                                                                           70
      c = fgetc(in);
      if(c == EOF)
        return(ZUENDE);                           /* vorzeitiges Dateiende */
      if((!strchr(erlaubte_zeichen, c)) && (!strchr(metazeichen, c))
          && (c != separator))
        return(CHAR_SUCHTEXT);           /* ungueltiges Zeichen im Suchtext */
      if(c != separator)
        suchtext[i++] = c;
     }
suchtext[i] = '\0';                           /* Suchstring abschliessen */      80

if(c != '$')
  return(NICHT_GETRENNT);                   /* noetiges Trennzeichen fehlt */
else
  fgetc(in);

fscanf(in, "%80[^\n]", dummy);                     /* zweite Kommentarzeile */
fgetc(in);
if(dummy[0] != '#')
  return(NO_COMMENT);                               /* Kommentarzeile fehlt */   90

aktuelle_position = ftell(in);     /* Groesse des zu durchsuchenden Textes */
fseek(in, 0L, SEEK_END);           /* ermitteln durch Laenge der Datei     */
endposition = ftell(in);

anzahl = endposition-aktuelle_position+1;
*text = (char *) malloc((anzahl) * sizeof(char));
fseek(in, aktuelle_position, SEEK_SET);  /* zurueck zur aktuellen Position */

if(!(*text))                                                                     100
  return(SPEICHER);

for(i=0; i<(anzahl-1); ++i)   /* Einlesen des zu durchsuchenden Textes mit */
   {                          /* Fehlerpruefung                            */
    c = fgetc(in);
    if(!strchr(erlaubte_zeichen, c))
      return(CHAR_TEXT);
    (*text)[i] = c;
   }
(*text)[i] = '\0';                            /* Textstring abschliessen */      110

return(0);
```

```
}


/***************************************************************************/
/* Funktion "ausgabe"                                                      */
/***************************************************************************/
/*                                                                         */
/* Zweck:                                                                  */
/* ------                                                                  */
/* Bei korrektem Ablauf des Algorithmus Ausgabe der Ergebnisse            */
/*                                                                         */
/* Eingabeparameter:                                                       */
/* -----------------                                                       */
/* const char *suchtext   Zeichenkette mit dem eingelesener Suchtext bzw.  */
/*                        dem Muster wie vom Benutzer vorgegeben           */
/* const char *text       Zeichenkette mit dem eingelesenen Text, der zu   */
/*                        durchsuchen war                                  */
/* loesung l              Durch den Such-Algorithmus ermittelte Loesung    */
/* FILE *out              Zeiger auf die Ausgabedatei oder die Standard-   */
/*                        ausgabe                                          */
/*                                                                         */
/* Ausgabeparameter:                                                       */
/* -----------------                                                       */
/* keine                                                                   */
/*                                                                         */
/* Rueckgabewert:                                                          */
/* --------------                                                          */
/* int rc                 Fehlercode                                       */
/***************************************************************************/
int ausgabe(const char *suchtext, const char *text, loesung l, FILE *out)
{
 fprintf(out, "Muster:\n\n");
 fprintf(out, "%s\n\n", suchtext);

 fprintf(out, "Text:\n\n");
 fprintf(out, "%s\n\n", text);

 if(l.anfang != -1)
   fprintf(out, "Treffer: Position %d, Laenge %d\n", l.anfang, l.laenge);
 else
   fprintf(out, "Kein Treffer!\n");

 return(0);
}



/***************************************************************************/
/* Funktion "fehlerausgabe"                                                */
/***************************************************************************/
/*                                                                         */
/* Zweck:                                                                  */
/* ------                                                                  */
/* Ausgabe von Fehlermeldungen auf der Standard-Fehlerausgabe             */
/*                                                                         */
/* Eingabeparameter:                                                       */
```

```
/* -----------------                                                    */     170
/* int fehler          Nummer des Fehlers (bezeichnet Art des Fehlers)  */
/*                                                                      */
/* Ausgabeparameter:                                                    */
/* -----------------                                                    */
/* keine                                                                */
/*                                                                      */
/* Rueckgabewert:                                                       */
/* --------------                                                       */
/* keine                                                                */
/************************************************************************/     180
void fehlerausgabe(int fehler)
{
 switch(fehler)
   {
    case(AUFRUF) :
    fprintf(stderr, "Programmaufruf: <programmname> <eingabedatei> "
                    "[ausgabedatei]\n");
    break;

    case(INDAT) :                                                               190
    fprintf(stderr, "Eingabedatei konnte nicht geoeffnet werden !\n");
    break;

    case(OUTDAT) :
    fprintf(stderr, "Ausgabedatei konnte nicht geoeffnet werden !\n");
    break;

    case(SPEICHER) :
    fprintf(stderr, "Kein Speicher zum Allozieren des Textstrings !\n");
    break;                                                                      200

    case(DAT_LEER) :
    fprintf(stderr, "Eingabedatei enthaelt keine verwertbaren Daten !\n");
    break;

    case(ZUENDE) :
    fprintf(stderr, "Vorzeitiges Ende der Eingabedatei !\n");
    break;

    case(NO_COMMENT) :                                                          210
    fprintf(stderr, "Kommentarzeichen fehlt !\n");
    break;

    case(NICHT_GETRENNT) :
    fprintf(stderr, "Suchtext und Text nicht ordnungsgemaess getrennt !\n");
    break;

    case(CHAR_SUCHTEXT) :
    fprintf(stderr, "Ungueltige Zeichen im Suchtext !\n");
    break;                                                                      220

    case(CHAR_TEXT) :
    fprintf(stderr, "Ungueltige Zeichen im Textstring !\n");
    break;

    case(SYNTAX_SUCHSTR) :
```

```
    fprintf(stderr, "Syntax des Suchstrings nicht korrekt !\n");
    break;

    case(FALSCH_IN_ZK) :                                                        230
    fprintf(stderr, "Nicht erlaubte Zeichen in einer Zeichenklasse !\n");
    break;

    case(DOPPELT_IN_ZK) :
    fprintf(stderr, "Suchtext zu lang wegen doppelten Zeichen in "
                    "Zeichenklasse !\n");

    case(LEERE_ZK) :
    fprintf(stderr, "Leere Zeichenklasse aufgetreten !\n");
    break;                                                                      240

    break;
   }
 return;
}
```

D.6 Header-Datei verarbeitung.h

```
#ifndef _VERARBEITUNG_H_INCLUDED_
  /****************************************************************************/
  /* Header-Datei verarbeitung.h                                              */
  /****************************************************************************/
  /*                                                                          */
  /* Aufgabe:                                                                 */
  /* --------                                                                 */
  /* Macht die globalen Datenstrukturen im Modul verarbeitung.c nutzbar und */
  /* die Schnittstelle der Funktion suche() dem Hauptprogramm bekannt.      */
  /*                                                                          */  10
  /* Enthaltene Prototypen:                                                   */
  /* ----------------------                                                   */
  /* extern loesung suche(const char *suchtext, const char *text);          */
  /*                                                                          */
  /* Enthaltene Makrodefinitionen:                                            */
  /* -----------------------------                                            */
  /* keine                                                                    */
  /****************************************************************************/

  #define _VERARBEITUNG_H_INCLUDED_                                               20
  #include "global_header.h"
  #include "myerrors.h"

  extern loesung suche(const char *suchtext, const char *text);

#endif  /* _VERARBEITUNG_H_INCLUDED_ */
```

D.7 Modul verarbeitung.c

```
/****************************************************************************/
/* Modul verarbeitung.c                                                     */
/****************************************************************************/
/*                                                                          */
/* Aufgabe:                                                                 */
/* --------                                                                 */
/* Dieses Modul enthaelt die Funktionen zum Suchen eines regulaeren         */
/* Ausdruckes in einem Text.                                                */
/*                                                                          */
/* Enthaltene Funktionen:                                                   */   10
/* ----------------------                                                   */
/* static bool zuweisen(muster *m, const char *c, char stern)               */
/* static void komplement(const char *c1, char *c2, const char *grund)      */
/* static int lese_zk(const char *suchtext, int *i,                         */
/*                    const char *erlaubte_zeichen, char *dummy)            */
/* static muster musteranalyse(const char *suchtext)                        */
/* static void suche_in_text(muster m, const char *text, int position,      */
/*                           int zustand, bool *treffer, int *len)          */
/* loesung suche(const char *suchtext, const char *text)                    */
/****************************************************************************/   20


#include "verarbeitung.h"



/****************************************************************************/
/* Funktion "zuweisen"                                                      */
/****************************************************************************/
/*                                                                          */   30
/* Zweck:                                                                   */
/* ------                                                                   */
/* Einen Bestandteil eines Musters belegen                                  */
/*                                                                          */
/* Eingabeparameter:                                                        */
/* -----------------                                                        */
/* const char *c         Passende Zeichen des zu belegenden Bestandteils    */
/* char stern            Naechstes Eingabezeichen; entscheidet, ob Wie-     */
/*                       derholung zulaessig ist oder nicht                 */
/*                                                                          */   40
/* Ausgabeparameter:                                                        */
/* -----------------                                                        */
/* muster *m             Zeiger auf das zu belegende Muster                 */
/*                                                                          */
/* Rueckgabewert:                                                           */
/* --------------                                                           */
/* bool weiter           Gibt an, ob ein weiteres Zeichen ueberlesen        */
/*                       werden muss                                        */
/****************************************************************************/
static bool zuweisen(muster *m, const char *c, char stern)                       50
{
 bool weiter;

 strcpy(m->teile[m->bestandteile].passende_zeichen, c);
 if(stern == '*')
```

```
   {
    m -> teile[m->bestandteile++].wiederholung = true;
    weiter = true;
   }
 else                                                                        60
   {
    m -> teile[m->bestandteile++].wiederholung = false;
    weiter = false;
   }

 if((m->teile[m->bestandteile-1].wiederholung) &&
    (m->teile[m->bestandteile-2].wiederholung) &&
    (!strcmp(m->teile[m->bestandteile-1].passende_zeichen,
             m->teile[m->bestandteile-2].passende_zeichen)))
    m->bestandteile--;                          /* keine neue Information */    70


 return(weiter);
}



/****************************************************************************/
/* Funktion "komplement"                                                    */
/****************************************************************************/    80
/*                                                                          */
/* Zweck:                                                                   */
/* ------                                                                   */
/* Das Komplement einer Zeichenkette in einer anderen Zeichenkette          */
/* ermitteln.                                                               */
/*                                                                          */
/* Eingabeparameter:                                                        */
/* -----------------                                                        */
/* const char *c1        Zeichenkette, deren Komplement ermittelt werden   */
/*                       soll                                               */    90
/* const char *grund     Zeichenkette, bezueglich der das Komplement von   */
/*                       c1 ermittelt werden soll                           */
/*                                                                          */
/* Ausgabeparameter:                                                        */
/* -----------------                                                        */
/* char *c2              Zeichenkette; enthaelt nach Ende der Funktion das */
/*                       Komplement von c1 in Grund                         */
/*                                                                          */
/* Rueckgabewert:                                                           */
/* --------------                                                           */    100
/* keiner                                                                   */
/****************************************************************************/
static void komplement(const char *c1, char *c2, const char *grund)
{
 int i;
 char c;
 int pos=0;

 for(i=0; i<ALPHA; ++i)
    {                                                                        110
     c = grund[i];
     if(!strchr(c1, c))
```

```
        c2[pos++] = c;
     }

 return;
}

                                                                              120
/***************************************************************************/
/* Funktion "lese_zk"                                                      */
/***************************************************************************/
/*                                                                         */
/* Zweck:                                                                  */
/* ------                                                                  */
/* Eine Zeichenklasse wird eingelesen                                      */
/*                                                                         */
/* Eingabeparameter:                                                       */
/* -----------------                                                       */   130
/* const char *suchtext   Zeichenkette, aus der die Zeichenklasse gelesen  */
/*                        werden soll                                      */
/* const char             Zeichenkette, die die Menge der zulaessigen      */
/* *erlaubte_zeichen      Zeichen der zu lesenden Klasse beinhaltet        */
/*                                                                         */
/* Ausgabeparameter:                                                       */
/* -----------------                                                       */
/* int *i                 Zeiger auf die aktuelle Position im Suchtext     */
/* char *dummy            Zeichenkette, auf die die Zeichenklasse          */
/*                        eingelesen wird.                                 */   140
/*                                                                         */
/* Rueckgabewert:                                                          */
/* --------------                                                          */
/* vom Typ int            Fehlercode                                       */
/***************************************************************************/
static int lese_zk(const char *suchtext, int *i,
                   const char *erlaubte_zeichen, char *dummy)
{
 int j=0;
 int c;                                                                          150

 while(((c=suchtext[++(*i)]) != ']') && (j<ALPHA))
      {
       if(!strchr(erlaubte_zeichen, c))
         return(FALSCH_IN_ZK);
       dummy[j++] = c;
       dummy[j] = '\0';
      }

 if(j==0)                                                                        160
   return(LEERE_ZK);
 if(c != ']')
   return(DOPPELT_IN_ZK);     /* Doppelte Zeichen muessen die Maximallaenge */
                              /* ueberschritten haben                       */
 return(0);
}
```

```
/****************************************************************************/
/* Funktion "musteranalyse"                                                 */
/****************************************************************************/
/*                                                                          */
/* Zweck:                                                                   */
/* ------                                                                   */
/* Eine Zeichenkette wird in Musterbestandteile zerlegt und das Ergebnis in */
/* der datenstruktur "muster" gespeichert.                                  */
/*                                                                          */
/* Eingabeparameter:                                                        */
/* -----------------                                                        */
/* const char *suchtext   Zeichenkette, die das Muster enthaelt             */
/*                                                                          */
/* Ausgabeparameter:                                                        */
/* -----------------                                                        */
/* keine                                                                    */
/*                                                                          */
/* Rueckgabewert:                                                           */
/* --------------                                                           */
/* muster m               Muster, welches aus der Zeichkette ermittelt      */
/*                        wurde (enthaelt ggfs. Fehlerinformationen)        */
/****************************************************************************/
static muster musteranalyse(const char *suchtext)
{
 muster m;
 const char erlaubte_zeichen[ALPHA+1] =
 "ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜabcdefghijklmnopqrstuvwxyzäöü"
 "0123456789 ()+-=\n.,!?:;";
 char aktuelles_zeichen;
 char dummy[ALPHA+1], dummy2[ALPHA+1];
 bool interpretation=true, weiter;
 int i=0,n, rc;

 m.bestandteile = 0;
 n = strlen(suchtext);
 while((i<n) && (m.bestandteile < MAXBESTAND))
      {
       aktuelles_zeichen = suchtext[i];
       if(interpretation)
         {
          if(strchr(erlaubte_zeichen, aktuelles_zeichen) &&
             (aktuelles_zeichen != '.'))     /* kein Metazeichen            */
            {
             dummy[0] = aktuelles_zeichen;   /* Nur aktuelles_zeichen passt */
             dummy[1] = '\0';
             weiter = zuweisen(&m, dummy, suchtext[++i]);
             if(weiter)
               ++i;
            }
          else
           if((aktuelles_zeichen != '[')  &&
              (aktuelles_zeichen != '\\') &&
              (aktuelles_zeichen != '.'))
              {
               m.bestandteile = SYNTAX_SUCHSTR;           /* Syntax-Fehler  */
               return(m);
              }
```

```
      else
        switch(aktuelles_zeichen)
          {
           case '\\' :            /* Keine Interpretation des naechsten */     230
                                  /* Zeichens                           */
           interpretation = false;
           ++i;
           break;

           case '.'  :                 /* Alle erlaubten Zeichen passen */
           weiter = zuweisen(&m, erlaubte_zeichen, suchtext[++i]);
           if(weiter)
             ++i;
           break;                                                               240

           case '['  :
           if(suchtext[i+1] != '^')                        /* Zeichenklasse */
             {
              rc = lese_zk(suchtext, &i, erlaubte_zeichen, dummy);
              if(rc)
                {
                 m.bestandteile = rc;
                 return(m);
                }                                                               250
              else
                {
                 weiter = zuweisen(&m, dummy, suchtext[++i]);
                 if(weiter)
                   i++;
                }
             }
           else                          /* Komplement der Zeichenklasse */
             {
              ++i;                                  /* Ueberlesen von '^' */    260
              rc = lese_zk(suchtext, &i, erlaubte_zeichen, dummy);
              if(rc)
                {
                 m.bestandteile = rc;
                 return(m);
                }
              else
                {
                 komplement(dummy, dummy2, erlaubte_zeichen);
                 weiter = zuweisen(&m, dummy2, suchtext[++i]);                  270
                 if(weiter)
                   i++;
                }
             }
           break;
          }
    }
    else     /* Keine Interpretation => nur das naechste Zeichen passt */
      {
       if(strchr(erlaubte_zeichen, aktuelles_zeichen))                          280
         {
          dummy[0] = aktuelles_zeichen;
          dummy[1] = '\0';
```

```
              weiter = zuweisen(&m, dummy, suchtext[++i]);
              if(weiter)
                i++;
              interpretation = true;
             }
           else
             {                                                              290
              m.bestandteile = SYNTAX_SUCHSTR;
              return(m);
             }
          }
     }
 return(m);
}

                                                                            300
/**************************************************************************/
/* Funktion "suche_in_text"                                               */
/**************************************************************************/
/*                                                                        */
/* Zweck:                                                                 */
/* ------                                                                 */
/* Backtracking-Algorithmus zur Suche eines regulaeren Ausdruckes in einem  */
/* Text, Simulation eines endlichen Automaten.                            */
/*                                                                        */
/* Eingabeparameter:                                                      */   310
/* -----------------                                                      */
/* muster m              Muster, nachdem gesucht werden soll              */
/* const char *text      Text, indem das Muster gesucht werden soll       */
/* int position          Position, an der textvergleich stattfindet       */
/* int zustand           Zustand des endlichen Automaten, Rekursionsstufe  */
/*                                                                        */
/* Ausgabeparameter:                                                      */
/* -----------------                                                      */
/* bool *treffer         Gibt an, ob das Muster gefunden werden konnte    */
/* int *len              Dient zur Berechnung der Laenge des gefundenen   */   320
/*                       Musters                                          */
/*                                                                        */
/* Rueckgabewert:                                                         */
/* --------------                                                         */
/* keiner                Die Ergebnisse werden ueber die Ausgabe-Parameter */
/*                       zurueckgegeben                                   */
/**************************************************************************/
static void suche_in_text(muster m, const char *text, int position,
                          int zustand, bool *treffer, int *len)
{                                                                           330
 if(zustand == m.bestandteile)                    /* Endzustand erreicht ? */
   {
    *treffer = true;
    if((position-1) > *len)
      *len = position - 1;
   }
 else
   if(m.teile[zustand].wiederholung)          /* "Sternzustand" ?          */
     {
      suche_in_text(m,text,position,zustand+1,treffer,len);                340
```

```
      if(text[position+1] &&
         strchr(m.teile[zustand].passende_zeichen, text[position]))
        suche_in_text(m,text,position+1,zustand,treffer, len);
     }
   else
     if(text[position+1] &&
        strchr(m.teile[zustand].passende_zeichen, text[position]))
       {
        suche_in_text(m,text,position+1,zustand+1,treffer, len);
       }                                                                    350
}



/***************************************************************************/
/* Funktion "suche"                                                        */
/***************************************************************************/
/*                                                                         */
/* Zweck:                                                                  */
/* ------                                                                  */     360
/* Hauptunterprogramm. Ruft suche_in_text solange fuer nachfolgende Text-  */
/* positionen auf, bis entweder das Muster gefunden oder das Ende des zu   */
/* durchsuchenden Textes erreicht wurde.                                   */
/*                                                                         */
/* Eingabeparameter:                                                       */
/* -----------------                                                       */
/* const char *suchtext   Zeichenkette, die das zu suchende Muster enthaelt */
/* const char *text       Zeichenkette, die den zu durchsuchenden Test      */
/*                        enthaelt                                         */
/*                                                                         */     370
/* Ausgabeparameter:                                                       */
/* -----------------                                                       */
/* keine                                                                   */
/*                                                                         */
/* Rueckgabewert:                                                          */
/* --------------                                                          */
/* loesung l              Datenstruktur mit zwei Komponenten, die entweder */
/*                        die Loesung oder Fehlerinformationen beinhalten  */
/***************************************************************************/
loesung suche(const char *suchtext, const char *text)                             380
{
 static muster m;
 static char vorheriger_suchtext[(ALPHA+4)*MAXBESTAND+1];
 int position = 0;
 int n, len;
 bool treffer = false;
 loesung l;

 if(strcmp(suchtext, vorheriger_suchtext))    /* Untersuchen, ob das ver-   */
   {                                          /* gangene Muster wiederver-  */     390
    m = musteranalyse(suchtext);              /* wendet werden kann.        */
    strcpy(vorheriger_suchtext, suchtext);
   }

 if(m.bestandteile < 0)
   {
    l.anfang = m.bestandteile;
```

```
    return(l);
   }
                                                                              400
 n = strlen(text);
 while(!treffer && (position < n))
      {
       len = 0;
       suche_in_text(m,text,position++,0,&treffer,&len);
      }

 if(treffer)
   {
    l.anfang = position;                    /* Zaehlung soll bei 1 beginnen */    410
    l.laenge = len-(position-1)+1;
   }

 else
   l.anfang = NICHT_GEFUNDEN;


 return(l);
}
```