Programmiertechnik II

Sommersemester 2020 – Aufgabenblatt 2

Hasso-Plattner-Institut
Fachgebiet Computergrafische Systeme

Anmerkungen zu Aufgabenblatt 2

Der zu verwendende Programmrahmen steht als Zip-Archiv `uebung2.zip` im Moodle unter der Adresse `https://moodle.hpi3d.de/course/view.php?id=137` zum Download zur Verfügung.

Aufgabe 2.1: WHILE-Programme (10 Punkte 2+1+2+5)

Algorithmen können in Form sogenannter LOOP- oder WHILE-Programme angegeben werden. Bearbeiten Sie die folgenden Aufgaben **schriftlich** unter Nutzung der in Abbildung 1 dargestellten Syntax. x_i und x_j bezeichnen Variablen, die an dieser Stelle eingesetzt werden, c einen konstanten Wert. „$P; P$“ ist die Ausführung zweier Teilprogramme hintereinander, „LOOP x_i DO P END“ steht für die x_i-fache Ausführung des Teilprogramms P.

$$
\begin{array}{rcl}
P & ::= & x_i := x_j + c \\
 & | & x_i := x_j - c \\
 & | & P; P \\
 & | & \text{LOOP } x_i \text{ DO } P \text{ END}
\end{array}
$$

(a) LOOP-Programm

$$
\begin{array}{rcl}
P & ::= & x_i := x_j + c \\
 & | & x_i := x_j - c \\
 & | & P; P \\
 & | & \text{LOOP } x_i \text{ DO } P \text{ END} \\
 & | & \text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}
\end{array}
$$

(b) WHILE-Programm

Abbildung 1: Syntax der LOOP- und WHILE-Programme

a) Begründen Sie, warum LOOP-Programme immer nach endlicher Zeit terminieren.

b) Geben Sie ein WHILE-Programm an, das niemals terminiert.

c) Zeigen Sie, wie ein LOOP-Programm durch ein WHILE-Programm (ohne Verwendung von LOOP) simuliert werden kann.

d) Geben Sie drei WHILE-Programme ohne Verwendung von LOOP an, die die folgenden Funktionen berechnen. Dafür darf davon ausgegangen werden, dass gilt $x, y \in \mathbb{N}, x \geq 0, y > 0$.

   i) $x + y$

   ii) $x \cdot y$

   iii) x^y

Hinweis: Zuweisungen von Konstanten dürfen als $x_i := c$ notiert werden. Geben Sie jeweils an, welche Variable das Ergebnis der Berechnung enthält, nachdem das Programm terminiert.

Aufgabe 2.2: DFA (5 Punkte [2+3])

Deterministische endliche Automaten wechseln ihren internen Zustand bei jeder als gültig definierten Eingabe in einen eindeutig bestimmten Folgezustand. Nutzen Sie die Datei `dfa.cpp` des Programmrahmens, um den in Abbildung 2 dargestellten Automaten zu simulieren.

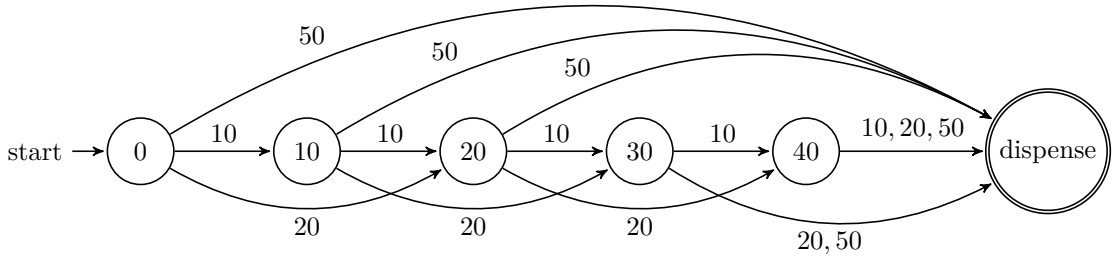

Abbildung 2: Getränkeautomat

An diesem Getränkeautomaten können Flaschen für einen Preis von jeweils 50 Cent erworben werden. Der Automat akzeptiert Münzen im Wert von 10, 20 oder 50 Cent. Sobald insgesamt mindestens 50 Cent eingeworfen wurden, gibt der Automat das Getränk aus und terminiert den Vorgang. Überzähliges Wechselgeld wird nicht zurückgezahlt. Nach Ausgabe eines Getränks wird der Automat in seinen Startzustand zurückgesetzt, ungültige Eingaben werden immer ignoriert.

Die Methode `step` erhält als Parameter den aktuellen Zustand des Automaten (*state*), den Betrag der eingegebenen Münze (*value*) und gibt daraufhin den neuen Zustand zurück. Verwenden Sie dafür keine Berechnungen, sondern ausschließlich `if`- und `switch`-Anweisungen auf den unveränderten Parametern. Der initiale Zustand des Automaten ist „0“ und der Automat erreicht den Zustand „dispense“, sobald das Getränk ausgegeben werden soll.

Hinweis: DFAs haben außer ihrem Zustand keine Möglichkeit zur Speicherung von Informationen. Sie dürfen aus diesem Grund keine Variablen anlegen, in denen Sie bspw. den Preis vorhalten. Der neue Zustand darf ausschließlich aus dem aktuellen Zustand und der Eingabe abgeleitet werden.

a) Vervollständigen Sie die Methode `step`, sodass der oben beschriebene Automat simuliert wird.

b) Erweitern Sie die Methode `stepExtended`, sodass der Automat zur Auswahl eines Getränkes als erste Eingabe die Zahl eines der folgenden Getränke mit jeweils definiertem Preis erfordert und das Getränk erst ausgibt, wenn die entsprechende Summe erreicht wurde:

   - 1: 50 Cent
   - 2: 80 Cent
   - 3: 100 Cent

   Der Aufruf des erweiterten Modus geschieht im mitgelieferten Programmrahmen, indem bei Programmstart ein beliebiges Kommandozeilenargument übergeben wird (z. B. `dfa.exe 1`)

Aufgabe 2.3: Reguläre Ausdrücke (6 Punkte)

Reguläre Ausdrücke ermöglichen es, Zeichenketten syntaktisch zu prüfen und zu parsen. In dieser Aufgabe sollen Sie **schriftlich** zu einem Adressinformationsmodell reguläre Ausdrücke ableiten. Eine Adresse bestehe im Folgenden aus diesen Elementen:

- Vorname (*Zeichenkette*)
- Nachname (*Zeichenkette*)
- Straße (*Zeichenkette*)
- Hausnummer (*Nummer ohne führende Null, auch zusammengesetzt oder in Verbindung mit maximal einem Buchstaben*)
- Postleitzahl (*5 Ziffern*)
- Ort (*Zeichenkette*)

Erstellen Sie geeignete reguläre Ausdrücke für die sechs Elemente der Adresse. Achten Sie darauf, dass in Ihren Ausdrücken mindestens die Beispieladressen aus Tabelle 1 komplett auf Korrektheit überprüft werden können und überlegen Sie sich Einschränkungen, die für die Zeichenketten in Deutschland üblicherweise gelten. Erklären Sie für jeden Ausdruck, welche Eingaben gültig sind und geben Sie jeweils ein korrektes und ein fehlerhaftes Beispiel an.

Vorname	**Nachname**	**Straße**	**Nr.**	**PLZ**	**Ort**
Max	Mustermann	Musterstraße	2a	12345	Berlin
Hasso	Plattner	Prof.-Dr.-Helmert-Str.	2-3	14482	Potsdam
Mark-Dieter	Kling	Paul-Heise-Weg	5	15711	Königs Wusterhausen
Bärbel	von Strauß	Am Häuschen	129	74172	Neckarsulm

Tabelle 1: Beispieladressen

Aufgabe 2.4: Verarbeitung von Textdateien (5 Punkte)

In dieser Aufgabe geht es darum, textuell gespeicherte Informationen strukturiert einzulesen und auszuwerten. Der im Programmrahmen mitgelieferte Datensatz *airports.dat* enthält Informationen zu Flughäfen. Das Modul `fileio.cpp` liest den Datensatz ein, wobei der Dateipfad und -name als Kommandozeilenargument übergeben wird.

a) Implementieren Sie in `readTokensAndLines` das zeilenweise Einlesen (mittels `std::getline`), wobei jede Zeile elementweise (getrennt durch einen Delimiter) zerlegt wird. Das Programm soll für jeden Flughafen den Namen und die zugehörige Zeitzone im Format `[Name] - [Database time zone]` auf `std::cout` ausgeben (z. B. `Mzuzu - Africa/Blantyre`).

b) Implementieren Sie die Funktion `isValueCorrect`, die einen String-Parameter auf korrekte Syntax überprüft. Es sollen die Spalten *ICAO*, *Altitude* und *DST* wie folgt auf Korrektheit überprüft werden:

- Für die Spalte *ICAO* ist ein drei- oder vierstelliger Code aus Großbuchstaben und Zahlen erlaubt oder ein leerer String.
- Für die Spalte *Altitude* ist ein nicht-negativer Höhenwert unter 30 000 Fuß als Ganzzahl erlaubt.
- Für die Spalte *DST* ist einer von sieben Eingabewerten erlaubt: E (Europe), A (US/ Canada), S (South America), O (Australia), Z (New Zealand), N (None) oder U (Unknown).

Falls durch das Programm ein syntaktischer Fehler im Eingabedatensatz entdeckt wird, soll eine entsprechende Warnung in eine Datei geschrieben werden. Hierzu sollen die gesamte Zeile des Eingabedatensatzes sowie der jeweilige ermittelte Fehler mittels `ofstream` in eine Logdatei `fileio.log` geschrieben werden. Nutzen Sie zur Überprüfung ausschließlich reguläre Ausdrücke (also keinen Größenvergleich von Zahlen) und die Funktion `regex_match` aus der Standardbibliothek `regex`.

Zusatzaufgabe: Entwerfen einer Klausuraufgabe (1 Bonuspunkt)

Formulieren Sie eine mögliche Klausuraufgabe zu den Inhalten des Kapitels 1 der Vorlesung. Die Aufgabe soll nicht einfach konkretes Wissen abfragen, sondern das Gesamtverständnis eines Sachverhalts und die Befähigung zur Übertragung auf neue Probleme zum Gegenstand haben.

Allgemeine Hinweise zur Bearbeitung und Abgabe

- Die Aufgaben können allein oder zu zweit bearbeitet werden. Je Gruppe ist nur eine Abgabe notwendig.
- Bonuspunkte können innerhalb des Übungsblattes verlorene Punkte ausgleichen. Sie sind nicht auf andere Übungsblätter übertragbar und werden nicht über die reguläre Punktzahl hinaus angerechnet.
- Bitte reichen Sie Ihre Lösungen bis spätestens **Mittwoch, den 20. Mai um 17:00 Uhr** ein.
- Die Implementierung kann auf einer üblichen Plattform (Windows, Linux, OS X) erfolgen, darf aber keine plattformspezifischen Elemente enthalten, d. h. die Implementierung soll plattformunabhängig entwickelt werden.
- Die Lösungen von theoretischen Aufgaben können als Textdatei oder PDF abgegeben werden. Benennen Sie die Dateien im Format `<Aufgabenblatt>_<Aufgabe>_<Teilaufgabe>`, bspw. „`2_1_d.txt`“ oder „`2_zusatz.txt`“.
- Bestehen weitere Fragen und Probleme, kontaktieren Sie den Übungsleiter oder nutzen Sie das Forum im Moodle.
- Archivieren Sie zur Abgabe Ihren bearbeiteten Programmrahmen und die Lösungen der theoretischen Aufgaben als Zip-Archiv und ergänzen Sie Ihre Namen im Bezeichner des Zip-Archivs im folgenden Format: **`uebung2_vorname1_nachname1_vorname2_nachname2.zip`**. Beachten Sie, dass dabei nur die vollständigen Lösungen sowie eventuelle Zusatzdaten gepackt werden (alle Dateien, die im gegebenen Programmrahmen vorhanden waren). Laden Sie keine Kompilate und temporären Dateien (*.obj, *.pdb, *.ilk, *.ncb, *.exe, etc.) hoch. Testen Sie vor dem Hochladen, ob die Abgabe fehlerfrei kompiliert und ausgeführt werden kann.
- Reichen Sie Ihr Zip-Archiv im Moodle ein:
`https://moodle.hpi3d.de/course/view.php?id=137`.