Programmiertechnik II

Sommersemester 2015 – 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=75` zum Download zur Verfügung.

Die Aufgabe 2.1 ist als schriftliche Aufgabe konzipiert. Die Aufgaben 2.2 und 2.3 sollen in den benannten und im Programmrahmen zumeist gekennzeichneten Bereichen implementiert werden. Bevor Sie mit der Lösung beginnen, lesen Sie das gesamte Aufgabenblatt und überprüfen Sie, ob der Programmrahmen auf Ihrem System fehlerfrei kompiliert, die Datensätze enthalten sind und sich alle Programme ausführen lassen. Wir wünschen viel Spaß und Erfolg bei der Bearbeitung der Aufgaben!

Aufgabe 2.1: Reguläre Ausdrücke (9 Punkte $^{6+3}$)

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 folgenden Elementen:

- Adresszeile 1: Vorname (*Zeichenkette*) Nachname (*Zeichenkette*)
- Adresszeile 2: Straße (*Zeichenkette*) Nummer (*auch in Verbindung mit einem Buchstaben, maximal 3 stellige Zahl und maximal ein Buchstabe*)
- Adresszeile 3: Postleitzahl (*positive 5-stellige Zahl*) Ort (*Zeichenkette*)

a) Erstellen Sie geeignete reguläre Ausdrücke für die sechs Elemente der drei Adresszeilen. Achten Sie darauf, dass in Ihren Ausdrücken mindestens die Beispieladressen aus Tabelle 1 komplett auf Korrektheit überprüft werden können.

b) Erweitern Sie die bisherigen regulären Ausdrücke in der Art, dass Sie folgende weitere Fälle abfangen können:

- Adresszusatz beginnend mit ”c/o” in separater Zeile
- Bis zu drei weitere Vornamen
- Doppelnamen in Nachnamen, getrennt durch einen Bindestrich

Vorname	Name	Straße	Nummer	Postleitzahl	Ort
Max	Mustermann	Musterstr.	2a	12345	Berlin
Mark-Dieter	Kling	Paul-Heise-Weg	5	15711	Königs Wusterhausen
Ines	Richter	Am Ufer	129	74172	Neckarsulm

Tabelle 1: Beispieladressen für Aufgabe 2.1a

Aufgabe 2.2: Verarbeitung von Textdateien (8 Punkte $^{3+5}$)

In dieser Aufgabe geht es darum, textuell gespeicherte Informationen struktkuriert einzulesen und auszuwerten. Der Datensatz *airports.txt*, enthält Informationen zu Flughäfen (Vergenwärtigen Sie sich den Datensatz in einem Texteditor). 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 auf `std::cout` ausgeben:

```
"[NAME] - [DATABASE Time zone]" -> "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 (die Überprüfung der übrigen Spalten ist optional):

- Für die Spalte *ICAO* ist ein exakt vierstelliger Buchstabencode komplett bestehend aus Großbuchstaben erlaubt oder ein leerer String.
- Für die Spalte *Altitude* ist ein positiver Höhenwert als Ganzzahl erlaubt (bis zu einem sinnvollen Maximum in Fuß).
- 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) or U (Unknown).

Falls durch das Programm ein syntaktischer Fehler im Eingabedatensatz entdeckt wird (...es sollten mindestens 3 gefunden werden), 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 reguläre Ausdrücke und die Funktion `regex_match()` aus der Standardbibliothek `<regex.h>`.

Aufgabe 2.3: Game of Life (9 Punkte 3+5+1)

In dieser Aufgabe soll eine einfache Variante des Game of Life implementiert werden. Zugrunde liegt dieser Simulation ein zweidimensionales Array, in dem spalten- und zeilenweise "Lebewesen"verzeichnet werden. Eine Zelle des Arrays ist somit entweder bevölkert oder unbevölkert. Beim Übergang von einem Zeitpunkt zum nächsten verändert sich die Population (d.h. der Zustand der einzelnen Zellen), wobei die Nachbarschaft der jeweils betrachteten Zelle einen wesentlichen Einfluss hat. Die Regeln sind wie folgt definiert:

- Ein Lebewesen entsteht in einer unbevölkerten Zelle, wenn diese genau drei bevölkerte Nachbarzellen hat.
- Ein Lebewesen mit keiner oder nur einer bevölkerten Nachbarzelle stirbt aufgrund von Einsamkeit.
- Ein Lebewesen mit vier oder mehr bevölkerten Nachbarzellen stirbt aufgrund einer zu hohen Population in seinem Umfeld.
- Ein Lebewesen mit zwei oder drei bevölkerten Nachbarzellen überlebt, weil sein Umfeld ausgewogen ist.

Im Programmrahmen übernimmt das Modul `game_of_life.cpp` die Auswertung der übergebenen Kommandozeilenargumente und initiiert die Simulation.

a) Import und Export: Erweitern Sie die gekennzeichneten Konstruktoren der Klasse `Raster`, um das zweidimensionale Array zu initialisieren. Der eine Konstruktor soll das Array auf Grundlage einer angegebenen Beispielgrafik (siehe hierzu die Bitmaps im Data-Ordner des Programmrahmens) initialisieren. Die Beispielgrafik kann über eine Kommandozeilenoption spezifiziert werden (z.B. `game_of_life -p ./example.bmp`). Der andere Konstruktor soll das Array auf Grundlage einer Zufallsfunktion befüllen. Die Größe des Arrays sowie die Wahrscheinlichkeit, mit der eine Zelle als bevölkert initialisiert wird, können über die Kommandozeile spezifiziert werden (z.B. `game_of_life -w 600 -h 400 -s 0.1`). Ein Wert von 0.5 führt beispielsweise dazu, dass die Hälfte der Zellen befüllt wird.

Implementieren Sie außerdem die Methode `save()` der Klasse `Raster`, um den Export der einzelnen Simulationsschritte als Bitmaps zu ermöglichen. Der finale Dateiname wird dieser Methode bereits übergeben. Das Ausgabeverzeichnis kann über die Kommandozeile spezifiziert werden (z.B. `game_of_life -p ./example.bmp -o output_directory/`).

b) Simulation: Implementieren Sie die Funktion `simulateNextState()`, um die Population zum nächsten Zeitpunkt der Simulation zu bestimmen. Ermitteln Sie hierzu die Anzahl an bevölkerten Zellen in einer 3x3-Nachbarschaft um die jeweilige Zelle. Wenden Sie anschließend die oben genannten Regeln an um den Folgezustand der Zelle zu bestimmen. Implementieren Sie die Funktion `neighborValue()` um den Wert einer angegebenen Zelle zu bestimmen. Fangen Sie ungültige Anfragen (z.B. Position außerhalb des Rasters) ab, indem Sie:

- die Zelle als unbevölkert angeben oder
- das Array als Torus-förmig ansehen (d.h. den Zustand der am gegenüberliegenden Rand liegenden Zelle zurückgeben).

Für die Simulation wird standardmäßig die erste Variante gewählt. Die zweite Variante kann jedoch über eine entsprechende Kommandozeilenoption (`-t 1`) gewählt werden. Die Anzahl an berechneten Simulationsschritten kann ebenfalls über die Kommandozeile definiert werden (z.B. `-i 20`).

c) Invasionssimulation: Implementieren Sie als Erweiterung die Funktion `simulateInvasion()`, die zu Beginn jedes Simulationsschritts den Zustand jeder Zelle mit einer angegebenen Wahrscheinlichkeit invertiert. Die Wahrscheinlichkeit, mit der eine Zelle invertiert wird, kann als Kommandozeilenargument spezifiziert werden (z.B. `-iv 0.005` für eine Wahrscheinlichkeit von 0.05%).

Zusatzaufgabe: Auswertung von Verzeichnisinformationen (1 Klausurpunkt)

In dieser Aufgaben soll ein Kommadozeilenwerkzeug ("dirinfo") entwickelt werden. Es durchsucht ein angegebenes Verzeichnis im Dateisystem rekursiv und ermittelt dabei, wie häufig Dateien mit einer bestimmten Endung auftreten und summiert deren Größe. Das Kommandozeilenwerkzeug soll diese Ergebnisse in eine CSV-Datei schreiben, sodass die Ergebnisse nachgenutzt werden können.

Der Programmrahmen definiert eine Funktion `traverseDirectory()`, die zu einem gegebenen Pfad die enthaltenen Verzeichniselemente (z.B. Dateien, Unterverzeichnisse, Links etc.) traversiert. Implementieren Sie folgende Funktionalität:

a) Erweitern Sie die Funktion `traverseDirectory()` so, dass rekursiv etwaige Unterverzeichnisse durchsucht werden. Sammeln Sie Informationen über die angetroffenen Dateien, kategorisiert anhand der Dateiendung. Implementieren Sie eine Funktionalität zum Schreiben dieser Ergebnisse in eine Datei im CSV-Format. Pro Dateikategorie sollen die folgenden Informationen gesammelt werden:

- Anzahl der Dateien
- Summierter Speicherbedarf (in Kilobytes)
- Liste von Dateipfaden zu den einzelnen Dateien der jeweiligen Kategorie (kodiert relativ zum Ausgangsverzeichnis)

Diese Informationen sollen dabei nur exportiert werden, wenn ein entsprechendes Kommandozeilenargument übergeben wurde. Implementieren Sie eine verwendungssichere Auswertung der Kommandozeilenargumente:

- `-fsize` : Fügt in die Ergebnisdatei den summierten Speicherbedarf als CSV-Spalte ein
- `-fnum` : Fügt in die Ergebnisdatei die Anzahl der ermittelten Dateien einer Dateikategorie als CSV-Spalte ein.
- `-files` : Fügt die Liste der einzelnen Dateien der Dateikategorie ein; jede Datei wird in eine einzelne CSV-Spalte geschrieben
- `-o` : Spezifiziert den Namen (und Pfad) der Ausgabedatei

Das Ausgangsverzeichnis und die Ausgabe-CSV-Datei sollen ebenfalls über die Kommandozeile spezifiziert werden. Ein beispielhafter Programmaufruf ist:
`dirinfo -fnum -fsize -files ./example_dir -o output.csv`.

b) Als Erweiterung soll mittels eines zusätzlichen Kommandozeilenarguments ein sogenanntes *Flattening* auf dem Ausgangsverzeichnis durchgeführt werden: Hierbei werden alle enthaltenen Dateien in ein gegebenes Ausgabeverzeichnis kopiert. Dabei wird der Pfad jeder Datei (relativ zum Ausgangsverzeichnis) als Präfix dem Dateinamen hinzugefügt. Die einzelnen Ordner werden dabei durch einen Unterstrich getrennt, z.B. wird der Dateipfad `my/dir/example.csv` auf den Dateinamen `my_dir_example.csv` abgebildet. Damit kann z.B. der Inhalt (Dateien) eines stark verschachtelten Verzeichnisse einfacher und nur einstufig gebündelt werden.

Werden mehrere Dateien auf den gleichen Dateinamen abgebildet, soll eine fortlaufende Nummer als Suffix hinzugefügt werden um einen eindeutigen Namen im "flachen"Verzeichnis immer sicherzustellen. Die fortlaufende Nummer soll für jeden abgebildeten Dateinamen separat verwaltet werden. So würden z.B. die Dateien `my/dir/example.csv` und `my_dir/example.csv` jeweils auf den Dateinamen `my_dir_example.csv` abgebildet. Um eindeutige Namen sicherzustellen, benennen Sie die zuletzt kopierte Datei stattdessen in `my_dir_example2.csv` um.

Da Standardoperationen für das Filesystem (z.B. Traversieren, Löschen oder Erzeugen von Verzeichnissen) derzeit noch nicht im Sprachstandard von C++ inkludiert sind, müssen Sie entsprechende Hilfsbibliotheken verwenden (z.B. `<dirent.h>` oder `<direct.h>`; siehe Programmrahmen). Die CMake-Konfigurationsdatei ist bereits entsprechend für die einzelnen Plattformen konfiguriert!

Zum Kopieren von Dateien mittels der C++-Standardbibliothek nutzen Sie geeignete Funktionen der `<fstream>`-Bibliothek.

Allgemeine Hinweise zur Bearbeitung und Abgabe

- Die Aufgaben sollen maximal zu zweit bearbeitet werden; Ausnahmen sind nicht vorgesehen. Je Gruppe ist nur eine Abgabe notwendig.
- Bitte reichen Sie Ihre Lösungen bis **Mittwoch, den 06. Mai um 13:00 Uhr** ein.
- Zur Prüfungszulassung muss ein Aufgabenblatt zumindest bearbeitet werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte bewertet werden. Aufgaben mit Klausurpunkten werden 1:1 berücksichtigt.
- Die Aufgaben können auf allen wesentlichen Plattformen (Windows, Linux, OS X) bearbeitet werden. Lesen Sie bei Fragen zum Kompilieren und Ausführen bitte genau die den Archiven jeweils beiligende README.TXT. Bestehen weitere Fragen und Probleme, kontaktieren Sie einen der Betreuer direkt oder nutzen Sie das Forum im Moodle.
- Die Durchsicht zur Bewertung kann sowohl auf Windows, Linux oder OS X erfolgen. Es wird folglich eine **plattformunabhängige Lösung** der Aufgaben erwartet.
- Archivieren (Zip) Sie zur Abgabe Ihren bearbeiteten Programmrahmen und ergänzen Sie Ihre Matrikelnummer(n) im Bezeichner des Zip-Archivs entsprechend im folgenden Format: **`uebung2_matrikNr1_matrikNr2.zip`**. Geben Sie die schriftlich bearbeitete Aufgabe 2.1 als Textdokument (.txt) oder PDF im Archiv hinzugefügt ab. Beachten Sie, dass dabei nur die vollständigen Quelltexte, die CMake-Konfiguration sowie eventuelle Zusatzdaten gepackt werden (alles was im gegebenen Programmrahmen entsprechend vorhanden ist). Senden Sie uns keinesfalls Kompilate und temporäre Dateien (*.obj, *.pdb, *.ilk, *.ncb etc.). Testen Sie vor dem Verschicken, ob die Projekte aus den Zipdateien fehlerfrei kompiliert und ausgeführt werden können.
- Reichen Sie Ihr Zip-Archiv im Moodle ein:
  `https://moodle.hpi3d.de/course/view.php?id=75`.