Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik
Institut für Informatik
Lehrstuhl für Digitale Bildverarbeitung

Dr. Paul Bodesheim

Übung zur Vorlesung
Informatik I (B.Sc. Physik)
SoSe 2020

Aufgabenblatt 4: Felder, Zeichenketten, Strukturen

Ausgabe: 30.06.2020 Abgabe: 07.07.2020

Die Besprechung der Übungsaufgaben findet eine Woche nach Abgabetermin als Video-Konferenz im Übungszeitraum statt. Beachten Sie die Hinweise zur Übung aus der Einführungsveranstaltung!

Aufgabe 1 Dominosteine (5 Punkte)

Dominosteine sind Spielsteine, auf denen jeweils zwei Zahlen von 0 bis 6 dargestellt sind, z.B. durch Punkte. Man darf Dominosteine so aneinanderlegen, dass jeweils gleiche Nummern nebeneinander liegen.

1	2		2	0		0	5		5	5

Ein Dominostein soll als ganze Zahl repräsentiert werden, indem die linke Zahl des Dominosteins als Zehnerstelle und die rechte Zahl des Dominosteins als Einerstelle dargestellt wird. Eine Kette von Dominosteinen kann dann in einem `vector<int>` abgelegt werden, sodass obige Kette als 12, 20, 5, 55 dargestellt wird.

Wir wollen für einen gegebenen int-Vektor überprüfen, ob die Zahlenfolge gültige Werte für Dominosteine enthält (entsprechend Punktzahlen von 0 bis 6) und ob immer gleiche Punktzahlen nebeneinander liegen. Schreiben Sie dazu folgende (Teil-)Funktionen:

- `int Left(int n)`
  Die Funktion ermittelt zu einer Zahl n, die einen Dominostein beschreibt, die linke Punktanzahl (Zehnerstelle). (`26 -> 2`)
- `int Right(int n)`
  Die Funktion ermittelt zu einer Zahl n, die einen Dominostein beschreibt, die rechte Punktanzahl (Einerstelle). (`26 -> 6`)
- `bool Valid(int n)`
  Die Funktion entscheidet, ob eine gegebene Punktanzahl zulässig ist (0 bis 6).
  (`5 -> true`, `7 -> false`)
- `bool Valid(vector<int> v)`
  Die Funktion entscheidet, ob der übergebene Vektor im oben genannten Sinne eine zulässige Kette von Dominosteinen repräsentiert.

Testen Sie die Funktionen in einem geeignetem Hauptprogramm. Die Funktionen bauen aufeinander auf und können sich gegenseitig aufrufen.

Hinweis: Funktionen zur Eingabe und formatierten Ausgabe von Dominostein-Ketten können für das Testen hilfreich sein.

Aufgabe 2 Adressfilter (5 Punkte)

Aus Angst vor einer Spam-Flut werden E-Mail-Adressen im Netz oft verfremdet bzw. ”verschlüsselt”, indem die Zeichen ’@’, ’.’ und ’-’ durch ”(at)”, ”(dot)” und ”(minus)” ersetzt werden.
Aus `noo@uni-jena.de` wird zum Beispiel `noo(at)uni(minus)jena(dot)de`

a) Schreiben Sie ein Programm, welches in einer Liste von eingegebenen E-Mail-Adressen die genannten Ersetzungen durchführt!

b) Schreiben Sie ein zweites Programm, welches eine Eingabe aus verfremdeten Adressen wieder ”entschlüsselt”!

Die Eingabe erfolgt in beiden Programmen (zeilenweise) von cin. Beide Programme sollen in einer Schleife Eingaben solange abfragen und transformieren, bis nichts (der Leerstring) eingegeben wird. Anschließend sollen die transformierten Eingaben ausgegeben werden.

Aufgabe 3 Fußball-Ergebnisse (5 Punkte)

Im Fußball (und anderen Sportarten) werden Spielergebnisse in der Form 3:4 dargestellt, wobei die Zahlen links und rechts des Doppelpunktes die geschossenen Tore der eigenen bzw. der fremden Mannschaft darstellen. Die Mannschaft mit den meisten Toren hat gewonnen und erhält (in der Regel) 3 Punkte, bei Gleichstand / Unentschieden erhalten beide Mannschaften einen Punkt. In einer Zeichenkette werden die Spiel-Ergebnisse einer Mannschaft in der Form ”3:4, 1:0, 3:3” dargestellt, wobei zwischen den Einzelergebnissen ein Komma und beliebig viele Leerzeichen stehen.

Schreiben Sie eine Funktion **score**, die aus einer derartigen Zeichenkette die Punktzahl der Mannschaft berechnet. Bei einer ungültigen Zeichenkette gebe die Funktion -1 zurück. Im genannten Beispiel ”3:4, 1:0, 3:3” wären das erste Spiel verloren (0 Punkte), das zweite Spiel gewonnen (3 Punkte) und das dritte Spiel unentschieden (1 Punkt) - **score** muss also 4 als Ergebnis liefern.

Testen Sie ihre Funktion mit einem Hauptprogramm, indem Sie beliebig viele Spielergebnisse eingeben können, für die die Punktzahl einer Mannschaft ermittelt und ausgegeben wird.

Um Zeichenketten (`string`*) eingeben zu können, die auch Leerzeichen enthalten können, benutzen Sie die Funktion* `getline`*:*

```
...
string str;
getline(cin, str);
...
```

Hinweise:

- *Wir wollen vereinfacht einstellige Toranzahlen voraussetzen.*
- *Eine Ziffer (*`char c`*) kann man mit* `c - '0'` *in eine Zahl umwandeln.*

Aus der Klasse **string** dürfen hier nur die Operatoren (=, +, [], <, ==, ...) und die Methoden **length() bzw. size()** verwendet werden. C-Funktionen für Zeichen (**char**) dürfen nicht verwendet werden.

Aufgabe 4 Telefonbuch (10 Punkte)

Programmieren Sie Datenstrukturen und Funktionen für ein (einfaches) Telefonbuch! In dem Telefonbuch soll ein Eintrag nur einen Namen (string) und eine Telefonnummer (string) enthalten.

- Definieren Sie eine Datenstruktur `Entry` für einen Telefonbucheintrag!

- Legen Sie ein Telefonbuch (als `vector<Entry>`) an!
- Schreiben Sie eine Funktion zum Einfügen eines Telefonbucheintrages!
  ```
  void insert(vector<Entry> &telefonbuch,
              const string &name,const string &nummer);
  ```
- Schreiben Sie eine Funktion zum Löschen eines Telefonbucheintrages anhand des Namens!
  ```
  void deleteByName(vector<Entry> &telefonbuch,const string &name);
  ```
- Schreiben Sie eine Funktion zum Löschen eines Telefonbucheintrages anhand der Nummer!
  ```
  void deleteByNumber(vector<Entry> &telefonbuch,const string &nummer);
  ```
- Schreiben Sie eine Funktion, die die Einträge eines Telefonbuchs ausgibt!
  ```
  void print(const vector<Entry> &telefonbuch);
  ```
- Schreiben Sie eine Funktion, die einen vorgegebenen Namen im Telefonbuch findet und die Nummer zurückgibt!
  ```
  string numberByName(vector<Entry> &telefonbuch,const string &name);
  ```
- Schreiben Sie eine Funktion, die eine vorgegebene Nummer im Telefonbuch findet und den Namen zurückgibt!
  ```
  string nameByNumber(vector<Entry> &telefonbuch,const string &nummer);
  ```
- Schreiben Sie eine Funktion, die zu einem vorgegebenen Namensanfang alle Einträge zurückgibt, deren Namenseinträge am Anfang mit dem gegebenen Wert übereinstimmen.
  ```
  vector<Entry> fuzzySearch(vector<Entry> &telefonbuch,
                            const string &namensanfang);
  ```
  Hinweis: Schreiben sie eine Hilfsfunktion, die testet, ob ein übergebener String den Anfang eines zweiten String darstellt!

In Ihrem Hauptprogramm soll zunächst ein leeres Telefonbuch angelegt werden. Bieten Sie dann dem Nutzer die Funktionen:

1. Einfügen
2. Anzeigen
3. Löschen eines Namens
4. Löschen einer Nummer
5. Nummer zu einem Namen suchen
6. Namen zu einer Nummer suchen
7. Beenden

an (z.B. in Form einer Menu-Auswahl durch Eingabe von Ziffern). Bei der Suche der Nummer zu einem Namen soll zunächst die exakte Suche verwendet werden. Wird kein Eintrag gefunden, so soll mittels der unscharfen Suche versucht werden, Einträge zu finden und die gefundenen Werte auszugeben.

Viel Erfolg!