Programmieren / Algorithmen und Datenstrukturen 1 - Praktikum 4

Michael Roth
michael.roth@h-da.de

Einleitung

Sie implementieren in diesem Praktikum eine elektronische Variante des Spiels „Mastermind“.
Eigentlich ist dies ein Spiel für zwei Spieler, bei der einer der Spieler einen **Code** notiert, und der andere Spieler diesen Code zu raten versucht. Der erste Spieler gibt dabei nach jedem Rateversuch des zweiten Spielers Hinweise in Form von schwarzen bzw. weißen Markierungen.

Einen **schwarzen** Marker erhält der Spieler für **jede** richtige Ziffer des Codes, welche an der richtigen Stelle steht.
Weiße Marker werden für jede Ziffer vergeben, die zwar im Code vorkommt, aber an einer anderen Stelle.

Typischerweise wird Mastermind mit einem vierstelligen Code aus sechs Farben gespielt, wobei jede Farbe nur maximal einmal vorkommen kann. Für dieses Praktikum werden anstelle von Farben Zahlen oder Buchstaben verwendet.

Hinweise zu den Spielregeln finden Sie bei Bedarf im Internet.

OpenSpace

Für eine Abgabe in openSpace müssen Sie die Vorgabedatei `functions.hpp` verwenden und dort die Funktionen implementieren. Vom System überprüft werden nur die drei Funktionen aus Abschnitt 1, das eigentliche Spiel wird vom System nicht überprüft. Dies soll während der Abnahme im Praktikum geschehen.

Für die Praktikumsgruppe bei Herrn Roth ist eine Abgabe über openSpace verpflichtend.

1 Hilfsfunktionen

Die folgenden Funktionen stellen Hilfen für die Implementierung des Masterminds Spiel zur Verfügung. Die Funktionen sind jeweils als **Template** zu verfassen, damit diese unabhängig vom gewählten Datentyp funktionieren.

Hinweise: Die Deklarationen sind bereits in der zur Verfügung gestellten Datei `functions.hpp` enthalten. Daher wird empfohlen, diese Datei zu verwenden, auch wenn eine Abgabe über openSpace nicht gemacht wird.
Templatefunktionen sollten, **im Gegensatz zu normalen Funktionen**, in der Header-Datei implementiert werden. Die Erweiterung `.hpp` wird normalerweise für Templates verwendet.

1.1 Funktion `isValidInput`

Die Funktion

```
template <typename T>
bool isValidInput(const vector<T>& guess, int len, T min, T max)
```

soll überprüfen, ob der eingegebene Code `guess` **gültig** ist. Die Funktion liefert `true`, falls:

- `guess` die richtige Länge `len` hat, **und**
- Jedes Element aus `guess` zwischen `min` und `max` liegt.

Ansonsten soll die Rückgabe `false` sein.

1.2 Funktion `black`

Die Funktion

```
template <typename T>
int black(const vector<T>& solution, const vector<T>& guess)
```

liefert die **Anzahl** an schwarzen Markern für das übergebene Pärchen aus zu ratendem Code `solution` und des Rateversuches `guess`. Ein schwarzer Marker wird dann erteilt, wenn Wert und Position einer Ziffer übereinstimmen.

Beispiele:

guess	solution	Anzahl Marker
1234	1234	4
4321	1234	0
1354	1234	2

1.3 Funktion `white`

Die Funktion

```
template <typename T>
int white(const vector<T>& solution, const vector<T>& guess)
```

liefert die **Anzahl** an weißen Markern für das übergebene Pärchen aus zu ratendem Code `solution` und des Rateversuches `guess`. Ein weißer Marker wird dann erteilt, wenn ein Wert aus `guess` in `solution` zwar vor kommt, aber nicht an der richtigen Stelle steht.

Beispiele:

guess	solution	Anzahl Marker
1234	1234	0
4321	1234	4
4351	1234	3

2 Klasse `Mastermind`

Erstellen Sie eine Klasse `Mastermind`, die ein Mastermind Spiel realisiert. Sie sollten dafür die in Abschnitt 1 auf Seite 1 geschriebenen Funktionen verwenden.

Das Spiel soll eine „Raterunde“ simulieren, wobei der Benutzer zunächst die zu ratende Kombination eingibt und anschließend sollen die Rateversuche gestartet werden. Wenn der Spieler nach acht Runden nicht das richtige Ergebnis geraten hat, ist das Spiel verloren.

Optionale Zusatzaufgabe: Schreiben Sie zwei verschiedene Klassen für das Mastermind Spiel, die jeweils andere Datentypen (beispielsweise Zeichen (`char`) und Zahlen (`int`)).