Übungsteil zu Funktionale Programmierung

Übersicht

Das beschriebene Format muss eingehalten werden damit die Abgabe positiv bewertet werden kann:

- Die Abgabe jeder Übung ist in **einer Datei** zu lösen (z.B. `Angabe1.hs`). Diese ist **in das Home-Verzeichnis am Abgabe-Server zu kopieren**. Der Name muss korrekt sein. Die Datei darf nicht in einem Unterordner abgelegt werden.
- **GHC 8.6.5** oder neuer ist zu verwenden.
- Jede Abgabedatei muss die **korrekte Moduldeklaration** haben.
- Aus technischen Gründen muss **jede geforderte Funktion, Konstante und Datentyp definiert** sein und **ihr Typ muss dem in der Angabe entsprechen**.
- Eigene Module können nicht definiert werden.

Empfehlung:

- Die Verwendung des Projekt Templates gemeinsam mit den Build-Systemen `cabal` oder `stack`.

Abgeben von Aufgaben

Abzugeben ist nur die Datei mit den gelösten Aufgaben. Dies kann, zum Beispiel, über `ssh` erfolgen mit dem Befehl:

```
scp Angabe1.hs f<matnr>@g0.complang.tuwien.ac.at:
```

Falls das Projekt Template verwendet wird ist der Befehl:

```
scp src/Angabe1.hs f<matnr>@g0.complang.tuwien.ac.at:
```

Am Server `g0` kann die Abgabe erneut getestet werden um sicherzustellen, dass alles funktioniert:

```
ghci Angabe1.hs
```

GHC 8.6.5

Die Abgaben werden automatisch mit GHC 8.6.5 ausgeführt.

Andere Versionen des GHC können für diese Übung auch verwendet werden. In diesen Fall muss bei der Abgabe überprüft werden, ob die abgegebene Datei vom GHC auf dem Server geladen werden kann (via `ghci Angabe1.hs`, siehe Abschnitt darüber).

Installation

Es werden mehrere Wege angeboten, GHC zu installieren. Es reicht eine der Anleitungen zu befolgen.

- Anleitung auf der offiziellen Webseite: https://www.haskell.org/platform/
- Installation via `ghcup`: https://www.haskell.org/ghcup/

- Installation via `stack`: https://www.haskellstack.org/
- Auf MacOs via Brew: `brew install ghc`
- Auf Linux mithilfe des integrierten Paket-Managers

Funktionsdefinitionen, Datentypen und Moduldeklaration

Aus technischen Gründen gibt es Anforderungen an den Inhalt der Abgabedateien. Falls diese nicht eingehalten werden, kann es sein sein, dass die gesamte Abgabe nicht bewertet werden kann.

1. Jede Abgabedatei muss richtig benannt werden.
2. Am Beginn der Datei muss das richtige Modul definiert sein. Etwa durch die Zeile `module Angabe1 where` am Anfang der Datei `Angabe1.hs`
3. Es ist notwendig alle geforderten Funktionen und Datentypen zu definieren, auch wenn diese nicht implementiert wurden!
4. Alle Funktionen müssen mit den richtigen Typdeklarationen versehen werden.
5. Alle Datentypen müssen wie in der Angabe definiert sein.
6. Eigene Module können nicht mittels `import` statements verwendet werden (das Bewertungssystem unterstützt nur die eine Abgabe-Datei).

Das mit jeder Angabe mitgelieferte Template soll dabei helfen diese Regeln einzuhalten.

Projekt Template

Wir stellen ein Projekt Template zur einfachen Organisation der Abgabedateien zur Verfügung. Die Verwendung des Templates ist nicht verpflichtend, hilft aber bei der Einhaltung der oben beschriebenen Vorgaben.

Das Template hat die folgende Struktur:

```
./
│   .gitignore
│   cabal.project
│   stack.yaml
│   template-fprog.cabal
│
├───src
│       Angabe1.hs
│       Angabe2.hs
│       Angabe3.lhs
│       Angabe4.lhs
│       Angabe5.hs
│       Angabe6.hs
│       Angabe7.hs
│
└───test
        TestSuite1.hs
        TestSuite2.hs
```

```
TestSuite3.hs
TestSuite4.hs
TestSuite5.hs
TestSuite6.hs
TestSuite7.hs
Main.hs
```

Für diesen Kurs reicht es aus, die Dateien in `src/` zu bearbeiten und hochzuladen. **Achtung**: Die Dateien sind direkt ins Home-Verzeichnis abzugeben und dürfen nicht in einem Unterordner liegen.

Projekt Dateien Übersicht

Die folgenden Dateien müssen nicht abgegeben werden und sollten auch nicht verändert werden:

- `cabal.project` beinhaltet Projekt-Metadaten für `cabal`.
- `template-fprog.cabal` beschreibt die Struktur und die Abhängigkeiten des Projekts.
- `stack.yaml` ist dem `cabal.project` ähnlich, aber spezifisch für `stack`.
- `.gitignore` definiert welche Dateien nicht von dem Versionierungstool `git` erfasst werden sollen.

Tests

Tests können in `test/Main.hs` oder in einer der `test/TestSuiteX.hs` Dateien geschrieben werden. Diese Datei muss nicht abgegeben werden, daher ist es nicht Pflicht diese Tests zu schreiben. Eine beliebige Test-Bibliothek kann verwendet werden, das Projekt Template verwendet tasty und tasty-hunit.

Nachdem die Abgabe bewertet wurde werden die Testdaten in einer Datei `TestSuiteX.hs` veröffentlicht. Diese Testdaten können in das Projekt Template kopiert werden um sie lokal auszuführen. Um, zum Beispiel, die Testdaten von `Angabe1.hs` vom Server zu kopieren kann der folgende Befehl ausgeführt werden:

```
scp f<matnr>@g0.complang.tuwien.ac.at:TestSuite1.hs test/TestSuite1.hs
```

Empfohlene Werkzeuge

Das Projekt Template kann mit den gängigen Werkzeugen `cabal` (Version: `3.0.0.0` oder neuer) und `stack` (Version: `2.0.0` oder neuer) gebaut und ausgeführt werden. Es reicht eines dieser Werkzeuge zu verwenden. Es wird davon abgeraten beide Systeme gleichzeitig zu verwenden.

Die Verwendung von `cabal` oder `stack` ist optional, hilft aber bei der Verwendung des Templates und bei der Einhaltung der Richtlinien für die Abgabe.

Verwendung von cabal

Dokumentation: https://cabal.readthedocs.io

```
# Cabal-Index erneuren
# (nur notwendig, wenn die cabal Installation schon länger zurück liegt)
cabal update
# erstelltes Programm in einer interaktiven Shell öffnen
ghci src/Angabe1.hs
# Alle Tests ausführen
cabal test
```

Achtung: Um `cabal` verwenden zu können muss `GHC` bereits installiert sein.

Verwendung von Stack

Dokumentation: https://docs.haskellstack.org

```
# Projekt bauen
stack build
# erstelltes Programm in einer interaktiven Shell öffnen
stack repl src/Angabe1.hs
# alle Tests ausführen
stack test
```