04_numbercheck: Fließkommazahlüberprüfung

Dipl.-Ing. Dr. Günter Kolousek

1 Allgemeines

- **Es gelten die gleichen Richtlinien wie beim ersten Beispiel!!!**

2 Aufgabenstellung

Schreibe ein C++ Programm `isnum`, das eine Zahl entweder als Kommandozeilenargument erhält oder von `stdin` liest (dann wird `-` als Argument mitgegeben) und überprüft, ob es sich um eine Fließkommazahl handelt. Wenn es sich um eine Fließkommazahl handelt, dann ist `true` auszugeben, anderenfalls `false` (also auf `stdout`).

Als Format für eine Fließkommazahl wird die "übliche" Notation verwendet. Beispiele hierfür sind:

- `123`, `0123`, `-123`, `+123`
- `0.123`, `.123`, `123.123`, `-0.123`, `+.123`, `123.`
- `+123.456e12`, `123.456e+12`, `-123.456e-12`, `123.e12`

Es darf allerdings keinerlei Bibliotheksfunktion verwendet werden. Die Aufgabe ist mit einem endlichen Automaten zu lösen (siehe Anleitung)!

Die Kommandozeilenschnittstelle hat folgendermaßen zu funktionieren:

```
$ isfp -h
Checks if word from stdin is correct floating point number
Usage: isfp [Options] [STDIN]

Positionals:
  STDIN TEXT                  stdin marker (must be '-')

Options:
  -h,--help                   Print this help message and exit
  -v,--value TEXT             The number to check
$ isfp -v 123
true
$ isfp -v -123
true
$ isfp -v -123.a
false
$ echo -123 | isfp -
true
```

```
$ echo -.123e-12 | isfp -
true
```

Weiters soll das Modul `numbercheck` mittels Unit-Test getestet werden!

3 Anleitung

Schreibe ein Programm entsprechend der Aufgabenstellung.

- Bevor wir zum "harten" Kodieren kommen, überdenken wir nochmals unsere derzeitige Build-Umgebung! Die header-only Bibliothek `CLI11` kopieren derzeit wir in jedem Beispiel in das lokale `include`-Verzeichnis und die header-only Bibliothek `fmt` haben wir mittels einem absoluten Pfad mittels einer `include_directories` Anweisung zum Projekt hinzugefügt. Analoges gilt für die header-only Bibliothek `doctest`.

  Beide Ansätze sind nicht sonderlich sinnvoll: Oftmaliges Kopieren und absolute Pfade...

  Deshalb stellen wir ab jetzt unser Beispiel (und auch die zukünftigen Beispiele) auf Meson-Option um. Siehe dir dazu den relevanten Abschnitt des Dokumentes `meson_tutorial` an! Dort findest du auch die Erklärung wieso wir den absoluten Pfad von einer Datei (`meson.build`) in eine andere Datei (`meson_options.txt`) verlagern.

- In diesem Beispiel musst du wenn ein '`-`' als Kommandozeilenargument mitgegeben wird von `stdin` lesen. Dafür gibt es in C++ die globale Variable `cin` (analog zu `cout` und `cerr`) und die Funktion `getline`.

- Das Programm soll entweder `true` oder `false` auf `stdout` ausgeben. Wird ein `bool` mittels `cout << true` ausgegeben, dann wird in diesem konkreten Fall `1` auf `stdout` erscheinen. Mittels des Manipulators `boolalpha` kann man dies umstellen, dass `true` erscheint.

- Das Überprüfen erfolgt mittels eines deterministischen endlichen Automaten in einem eigenen header-only Modul `numbercheck` (also nur eine Datei `numbercheck.h`) und in einem Namensraum `numbercheck`.

  Schreibe dafür eine Klasse `FloatingPointChecker`, der über eine Methode `bool check(string)` verfügt. Das ist zwar in diesem Fall nicht so sinnvoll, da die Methoden "relativ" groß werden und in einer Headerdatei aber implizit als `inline` markiert sind, aber für das Kennenlernen dieses Ansatzes ist es durchaus in Ordnung. Setze `public:` und `private:` ein.

  Verwende für die Zustände in deinem Automaten ein `enum class` und überlege dir welche Unterschiede es zu einem "normalen" `enum` und einer Implementierung auf Basis von `char` gibt.

  Aufgrund unserer `-Werror`-Projekteinstellung wirst du ein Attribut `[[fallthrough]]` mit einer leeren Anweisung (also `[[fallthrough]];`) benötigen!

  Verwende hierfür eine switch-basierte Implementierung in einer Klasse `FloatingPointChecker`.

  Alternativ könnte man auch einen Tabellen-basierten Automaten implementieren. Das ist eigentlich weniger Aufwand und außerdem flexibler, aber eben etwas weniger effizient.

- Die Unit-Tests sollen jetzt schon etwas ausgefeilter ausfallen. Implementiere daher die folgenden `TEST_CASE`s:
  - correct integer numbers without exponent
  - correct integer numbers with exponent
  - correct floating point numbers without exponent
  - correct floating point numbers with exponent
  - false numbers

Du solltest auf mindestens 25 `CHECK`s kommen.

4 Übungszweck dieses Beispiels

- `CLI11`
- Meson-Optionen und `meson_options.txt`
- `cin`, `getline()`, `boolalpha`
- Lesen von `cin` und Verkettung zweier Prozesses mittels einer Pipe
- `[[fallthrough]]` Attribut
- Entwurf endlicher Automaten
- Implementierung eines `switch`-basierten DEA
- `doctest`
- Einfache Klasse implementieren. `public:` und `private:`
- `enum class`