02_prime_factorization: Primfaktorenzerlegung

Dipl.-Ing. Dr. Günter Kolousek

1 Allgemeines

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

2 Aufgabenstellung

Schreibe ein C++ Programm `primes`, das eine Zahl (`unsigned long long`) auf der Kommandozeile erhält und die Primfaktoren auf der Kommandozeile ausgibt.

Die Kommandozeilenschnittstelle hat folgendermaßen zu funktionieren:

```
$ primes -h
Calculates the prime factors of the given number
Usage: primes [Options] NUMBER

Positionals:
  NUMBER UINT REQUIRED        The number

Options:
  -h,--help                   Print this help message and exit
$ primes 100a
Could not convert: NUMBER = 100a
Run with --help for more information.
$ primes 100
2 2 5 5
```

Weiters sind dafür Unit-Tests zu implementieren, die folgendermaßen gestartet werden:

```
$ ninja test
```

3 Anleitung

Schreibe ein Programm entsprechend der Aufgabenstellung.

- Es soll jetzt `meson` zur Generierung der Builddateien und `ninja` zum eigentlichen Übersetzen verwendet werden. Die Sourcedateien kommen in ein eigenes Unterverzeichnis `src`, die Headerdateien in ein eigenes Unterverzeichnis `include` und die erzeugten Dateien in das Unterverzeichnis `build`. Überfliege dazu das Tutorial `meson_tutorial` und schaue dir den Anfang des Dokumentes mit den relevanten Abschnitten *genau* an!

  Wichtig:

  - Als Sprachstandard verwenden wir in den Übungen `c++2a`!
  - Der Warnlevel wird mit 3 festgelegt!
  - Jede Warnung wird als Fehler betrachtet!

- Die Behandlung der Kommandozeilenarguemente wird in einem eigenen Modul `main` durchgeführt. Wir sehen, dass die eigentliche Verarbeitung einfacher ist als beim ersten Beispiel. Trotzdem ist es mühsam und langweilig immer die gleiche Logik zu programmieren. Deshalb verwenden wir jetzt die header-only Bibliothek `CLI11`. Am einfachsten ist diese zu verwenden, wenn einfach die zur Verfügung gestellte Headerdatei `CLI11.hpp` in das Verzeichnis `include` kopiert wird. Das ist zwar aus verschiedensten Gründen nicht optimal (mehrere Projekte, Aktualität), aber vorerst am einfachsten.

  Auftrag: Studiere die Dokumentation von `CLI11`, sodass du in der Lage bist einfache Kommandozeilenverarbeitung mit diesem Tool zu implementieren.

  Beachte wie "Options" in meiner Beispielsausgabe geschrieben wurde. Das kann mittels Formatierung der Hilfeausgabe geändert werden.

- Die Funktionalität zur Berechnung der Primfaktoren soll in einem eigenen Modul `primes` entwickelt werden. Dafür ist sowohl eine `.cpp` als auch eine `.h` Datei zu schreiben. Das ist ab jetzt Standard!

  Anstatt des Header-Guard kann auch eine `#pragma` Direktive verwendet werden. Eine solche Preprozessordirektive ist eine Anweisung an den Präprozessor, die jedoch nicht standardisiert ist. Das bedeutet, dass diese nicht von jedem Präprozessor verstanden wird.

  In unserem konkreten Fall sieht das so aus, dass wir den Header-Guard einfach durch

  ```
  #pragma once
  ```

  am Anfang der Headerdatei ersetzen können. Alle gängigen Präprozessoren verstehen dies und somit können wir davon Gebrauch machen. Ist einfacher, kürzer und weniger feheranfällig.

- Bis jetzt haben wir `unsigned long long` verwendet. Der C++ Standard gibt allerdings keine konkrete Angaben über die Größe einer derartigen Zahl an. Statt dessen gibt es einen Typ `uint64_t` von dem wir sicher sein können, dass es sich um eine vorzeichenlose ganze Zahl mit 64 Bits handelt. Entsprechender Header ist zu inkludieren!

- Diese Funktionalität des Moduls `primes` soll in einem eigenen Namensraum `prime_factorization` entwickelt werden. Dazu ist eine Funktion `calculate_primes` zu schreiben, die eine Zahl vom Typ `uint64_t` bekommt und einen `std::vector` von `uint64_t` Zahlen zurückliefert. Auch der Header für `vector` ist zu inkludieren.

- Der Algorithmus zur Zerlegung der Zahl `n` in die Primfaktoren kann folgendermaßen verbal formuliert werden:

  1. Solange `n` durch 2 geteilt werden kann, ist 2 ein Primfaktor und wird zum Ergebnis hinzugefügt, während `n / 2` zum neuen `n` wird.
  2. Schleife über alle `i` von 3 bis zur Quadratwurzel von `n` in Zweierschritten: Solange `n` durch `i` geteilt werden kann, `i` zum Ergebnis hinzufügen, während `n / i` zum neuen `n` wird.
  3. Wenn `n` größer als 2, dann ist `n` ebenfalls zum Ergebnis hinzuzufügen.

- Die Quadratwurzel kann mittels `std::sqrt` berechnet werden.

- Zur Ausgabe der Primfaktoren soll eine range-basierte `for`-Schleife verwendet werden.

4 Übungszweck dieses Beispiels

- `meson` und `ninja`
- `CLI11`
- `unsigned long long` vs. `uint64_t`
- Implementierung des Algorithmus zum Ermitteln der Primfaktoren einer Zahl
- `std::vector`
- `std::sqrt` und `<cmath>`
- range-basierte `for`-Schleife
- `#pragma once`