Beispiel `02_cppintro2`

Dr. Günter Kolousek

1. September 2017

1 Allgemeines

- Im ersten Beispiel gibt es genaue Anweisungen zum Aufbau und der Durchführung eines Beispiels. Bei Bedarf nochmals durchlesen!
- In diesem Sinne ist jetzt ein neues Verzeichnis `02_cppintro2` anzulegen.

2 Aufgabenstellung

Es ist ein `C++` Programm `primes` zu entwickeln, das überprüft, ob es sich bei gegebene Zahlen um Primzahlen handelt und weiters alle Primzahlen bis zu einer gegebenen Obergrenze ermittelt.

3 Anleitung

1. Schreibe ein Programm `primes`, das für eine auf der Kommandozeile übergebene Zahl n ($2 \leq n \leq 2.000.000.000$) überprüft, ob es sich um eine Primzahl handelt.

   Um aus einem Kommandozeilenparameter, der ein `char*` ist eine Zahl zu machen, verwende eine der Funktionen: `stoi`, `stol`, `stoll`, `stoul`, `stoull`, `stof`, `stod`, `stold`. Die Dokumentation dazu findest du auf `http://cppreference.com`. Diese Funktionen befinden sich im Namensraum `std`, wie alle anderen Funktionen und Typen der Standardbibliothek.

   Schreibe dazu eine eigene Funktion `isprime(n)`, die für eine übergebene Zahl `n` zurückliefert, ob diese eine Primzahl ist oder nicht.

   Und hier noch ein paar Hinweise:

   - Eine Primzahl ist eine Zahl größer als 1, die nur durch sich selbst und durch 1 ganzzahlig teilbar ist.
   - Die einzige gerade Primzahl ist 2.
   - Alle ungeraden Zahlen von 3 bis $\sqrt{n}$ sind zu testen, ob sie Teiler von n sind.

Erweitere jetzt das Programm, sodass eine ”usage” Meldung ausgegeben wird, wenn entweder kein Parameter mitgegeben wird oder wenn keine gültige Zahl als Parameter mitgegeben wird:

```
$ primes
Please provide a natural number (2 <= number <= 2.000.000.000)
usage: primes <number>
$

$ primes 1
Please provide a natural number (2 <= number <= 2.000.000.000)
usage: primes <number>
$
```

Auch die nachfolgende Verwendung sollte zu einer Fehlermeldung führen (konsultiere dazu die Dokumentation, im speziellen die Beispiele in der Dokumentation und auch den zweiten optionalen Parameter):

```
$ primes 1a
Please provide a natural number (2 <= number <= 2.000.000.000)
usage: primes <number>
$
```

Und so könnte es richtig aussehen:

```
$ primes 7
7 is a prime number
$
$ primes 4
4 is NOT a prime number
$
```

2. In Abhängigkeit der Größe des Zahlenbereiches muss immer der richtige Typ gewählt werden. Erweitere das Programm nochmals, dass Zahlen bis zu 10^{15} getestet werden können! Es ist nur eine kleine Änderung.

   Die Größe der einzelnen Bereiche kannst du dir folgendermaßen ermitteln, wobei der Typ des Templateparameters entsprechend zu ersetzen ist:

```
#include <limits>

cout << numeric_limits<double>::lowest() << endl;
cout << numeric_limits<double>::min() << endl;
cout << numeric_limits<double>::max() << endl;
```

Klarerweise verwenden wir keinen Datentyp `double`…

3. Erweitere nochmals, dass die Kommandozeilenschnittstelle jetzt in etwa folgendermaßen aufgebaut ist:

```
$ primes
Please provide a command!
usage: primes <command> <number>

Commands:
  test ... test if the given number is prime
  sieve ... lists all prime numbers up to the given number
$
```

Es muss aber auch ein korrektes Kommando eingegeben werden:

```
$ primes xxx
Please provide a correct command!
usage: primes <command> <number>

Commands:
  test ... test if the given number is prime
  sieve ... lists all prime numbers up to the given number
$
```

Auch wenn ein richtiges Kommando eingegeben wird, dann muss auch noch eine Zahl eingegeben werden:

```
$ primes test
Please provide a natural number (2 <= number <= 2.000.000.000)
usage: primes <command> <number>

Commands:
  test ... test if the given number is prime
  sieve ... lists all prime numbers up to the given number
$
```

Jetzt sollte das Programm auch richtig für das Kommando ”test” funktionieren:

```
$ primes test 1
Please provide a natural number (2 <= number <= 2.000.000.000)
usage: primes <command> <number>
```

```
Commands:
  test ... test if the given number is prime
  sieve ... lists all prime numbers up to the given number
$
$ primes test 7
7 is a prime number
$
```

- Implementiere jetzt eine Funktion `sieve_and_print(n)`, die alle Primzahlen bis zur übergebenen Zahl `n` ermittelt und auf der Konsole zeilenweise ausgibt. Die Funktion soll für alle Zahlen zwischen 2 und 5.000.000 funktionieren.

  Verwende dazu das Sieb des Eratosthenes!

  Hinweise:

  - Für das Sieb des Erathostenes muss man sich für jede Zahl merken, ob diese prim ist oder nicht. Jetzt stellt sich die Frage wie man dies implementiert. Einen normalen `bool` zu verwenden ist normalerweise nicht sinnvoll, da dieser aus Performancegründen in der Regel in einem Wort abgespeichert wird. Ein `char` zu verwenden ist schon sinnvoller, da dieser normalerweise nur ein Byte in Anspruch nimmt.

    Verwende trotzdem einen `vector<bool>`, da dieser "speziell" ist und nicht für jeden booleschen Wert ein Wort sondern nur ein Bit verwendet!
  - Entferne zuerst alle Vielfachen von 2, dann Vielfache der übrigen Zahlen 3,5,7,...

4 Übungszweck dieses Beispiels

- Kommandozeilenverarbeitung
- Implementierung von Algorithmen
- Verwendung von `vector`
- Schreiben einfacher Funktionen