01_gcd: Größter gemeinsamer Teiler

Dipl.-Ing. Dr. Günter Kolousek

1 Allgemeines

- Drucke dieses Dokument **nicht** aus!
- Halte **unbedingt** die Coding Conventions ein! Zu finden am ifssh!
- Lege erstmalig ein Verzeichnis mit deinem Nachnamen und deiner Matrikelnummer in der Form `<nachname>_<matrikelnummer>` an, also z.B. `mustermann_i15999`.
- In diesem Verzeichnis wird ein Mercurial-Repository angelegt!
- **Jedes** Beispiel ist in einem Unterverzeichnis `<beispielnummer_beispielname>` von dem vorhergehend angelegten Verzeichnis zu speichern! Für das aktuelle Beispiel heißt dieses Verzeichnis also `01_gcd`.
- Es müssen **ausreichend** Commits verfasst werden (siehe Foliensatz `revision_control_mercurial`).
- Als Betriebssystem wird in diesem Jahr **ausschließlich** Linux verwendet.
- Als Compiler wird ausschließlich g++ ab der Version 9.1 verwendet.

2 Aufgabenstellung

Schreibe ein C++ Programm `gcd`, das aus zwei Zahlen (`unsigned long long`) den größten gemeinsamen Teiler ermittelt (siehe Foliensatz `recursion`).

Die Kommandozeilenschnittstelle hat folgendermaßen zu funktionieren:

```
$ gcd -h
Computes the greatest common divisor of two numbers
Usage: gcd [Options] FIRST SECOND

Positionals:
  FIRST UINT REQUIRED       First number
  SECOND UINT REQUIRED      Second number

Options:
  -h,--help                 Print this help message and exit
$ gcd
exactly 2 arguments needed, but 0 given
Run with --help for more information.
$ gcd x
SECOND is required
```

```
Run with --help for more information.
$ gcd x y
Could not convert: FIRST = x
Run with --help for more information.
$ gcd 1 y
Could not convert: SECOND = y
Run with --help for more information.
$ gcd 366 60
computed iterative: 6
computed recursive: 6
$ gcd 111111111111111111111111111 60
FIRST has to be less or equal to 18446744073709551615: FIRST = 111111111111111111111111111
Run with --help for more information
```

D.h., das Programm soll genau so funktionieren und auch genau so aufgerufen werden können. Das bedeutet, dass der Pfad `.` in der Umgebungsvariable `PATH` enthalten sein muss (kein Problem für einen normalen Benutzer).

3 Anleitung

Schreibe ein Programm entsprechend der Aufgabenstellung. Es ist *vorerst* händisch zu übersetzen.

- Wiederhole die Foliensätze `codingconventions`, `os`, `linux` und `revision_control_mercurial`, `recursion`!!!
- Die Funktionalität zur Berechnung des größten gemeinsamen Teilers soll in einem eigenen Modul `gcd` entwickelt werden. Dafür ist sowohl eine `.cpp` als auch eine `.h` Datei zu schreiben. Die Headerdatei ist mit einem Guard abzusichern. Nicht vergessen: Die Headerdatei ist auch in der `.cpp` Datei des Moduls zu inkludieren, sonst kann der Compiler nicht die korrekte Verwendung überprüfen (→ Linker!).
- Diese Funktionalität des Moduls `gcd` soll in einem eigenen Namensraum `greatest_common_divisor` entwickelt werden. Das funktioniert in folgender Art und Weise:

  ```
  namespace xxx {
     xxx
  }
  ```

- Die Behandlung der Kommandozeilenarguemente wird in einem eigenen Modul `main` durchgeführt. Nicht vergessen: Zuerst Headerdatei des Moduls inkludieren, dann eigene Headerdateien, dann Headerdateien der verwendeten Bibliotheken und zum Schluss die Systemheaderdateien inkludieren.
- Zur Umwandlung der Strings in ganze Zahlen soll `stoull` verwendet werden. Exceptions abfangen und Position abfragen:

  ```
  size_t pos;
  string arg;
  try {
      res = stoull(arg, &pos);
  } catch (xxx&) {
     xxx
  } xxx
  if (pos != arg.si....)
  ```

- Im den `.cpp` Dateien *darf* ein `using namespace std;` verwendet werden, in den Headerdateien *niemals*! `using namespace` immer *nach* den `#include`-Direktiven!

- Entwickle sowohl eine rekursive als auch eine iterative Lösung, die jeweils in einem eigenen Unternamensraum von `greatest_common_divisor` verfasst werden sollen.
- Schreibe eine eigene freie Funktion `void usage(string msg="")`, die den Hilfetext entsprechend auf dem Kanal `stderr` ausgibt und danach den Prozess mit einem Exit-Code von 1 beendet (→ `exit()`).
- Der eigentliche Hilfetext soll in einer eigenen globalen "Variable"
  `constexpr string_view usage_msg{"Computes...` gespeichert werden.
  - Für was steht `constexpr`? Das steht für "constant expression" und bedeutet, dass der Compiler den Ausdruck auswerten kann.
  - Für was steht `string_view`? Schaue in der *cppreference* nach. Dort findest du:

    > The class template basic_string_view describes an object that can refer to a constant contiguous sequence of char-like objects with the first element of the sequence at position zero.

  Für den Text zur Anzeige der weiteren Hilfestellung `Run with --help...` verwende ebenfalls so eine "Variable".
- Es kann durchaus sinnvoll sein die freie Funktion `to_string()` aus der Standardbibliothek zu verwenden, die eine Zahl zu einem String konvertiert. RTFM!
- Den maximalen Wert für einen `unsigned long long` erhältst du vom Template `numeric_limits` aus dem Header `limits`. Prinzipiell auch RTFM, aber als Service meinerseits: `numeric_limits<unsigned long long>::max())`

4 Übungszweck dieses Beispiels

- Versionsverwaltung `mercurial` verwenden
- Einfaches C++ Programm mit einfachen und zusammengesetzten Anweisungen
- Übersetzen von der Kommandozeile
- Exit-Code setzen: `return` vs. `exit()`
- Implementieren des Algorithmus für den größten gemeinsamen Teilers (rekursiv und iterativ)
- Implementieren eines einfachen Moduls in C++
- `using namespace` einsetzen (using Direktive)
- Notwendigkeit für Header-Guards erkennen
- Verarbeitung und Überprüfung der Kommandozeilenargumente inkl. Ausgabe einer entsprechende Hilfe- und Fehlerausgaben
- Einfache Ausgabe: `cout`, `cerr`, `<<`, `endl`, `\n`
- Umwandeln von String zu Zahl: `stoull`,...
- Umwandeln einer Zahl zu String: `to_string`
- Abfangen von Exceptions
- Einfache Namensräume und Unternamensräume anlegen
- Einfache Verwendung von C-Stringliteralen, `string` und `string_view`
- Dokumentation `cppreference` verwenden