03_sorting: Sortieren

Dipl.-Ing. Dr. Günter Kolousek

1 Allgemeines

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

2 Aufgabenstellung

Schreibe ein C++ Programm `sortit`, das eine beliebige Sequenz von ganzen Zahlen (`int`) auf der Kommandozeile erhält und die sortiert Sequenz auf der Kommandozeile ausgibt.

Die Kommandozeilenschnittstelle hat folgendermaßen zu funktionieren:

```
$ sortit -h
Sorts the given sequence of numbers
Usage: sortit [Options] NUMBERS...

Positionals:
  NUMBERS INT ... REQUIRED    The numbers to sort

Options:
  -h,--help                   Print this help message and exit
[Option Group: algorithm]
  Choose the preferred sorting algorithm
  [At most 1 of the following options are allowed]
  Options:
    -b,--bubble                 Use bubblesort
    -i,--insertion              Use insertion sort (default)
    -s,--selection              Use selection sort
$ sortit 1 3 5 7 6 4 2 0
{0, 1, 2, 3, 4, 5, 6, 7}
$ sortit -i 1 3 5 7 6 4 2 0
{0, 1, 2, 3, 4, 5, 6, 7}
```

Weiters soll das Modul zum Sortieren mittels Unit-Test getestet werden.

3 Anleitung

Schreibe ein Programm entsprechend der Aufgabenstellung.

- Ab jetzt soll `meson` zur Generierung der Builddateien und `ninja` zum eigentlichen Übersetzen verwendet werden!

- Das Sortieren erfolgt in einem eigenen Modul `sorting` und in einem Namensraum `sorting`. Verwende in der Datei `sorting.cpp` jetzt kein Schlüsselwort `namespace`, sondern verwende den scope resolution operator.

  Der Prototyp für jede Sortierfunktion soll folgendermaßen gestaltet sein:

  ```
  void xxx_sort(unsigned int n, int numbers[]);
  ```

  Das ist zwar nicht sonderlich sinnvoll, aber es geht darum nicht nur mit Instanzen von `vector` arbeiten zu können, sondern auch mit rohen Feldern (raw arrays). Damit lernen wir auch wie wir von einem `vector` zu dem zugrundeliegenden Array kommen (`size()` und `data()`!).

- Die Ausgabe könnten wir natürlich wieder mit einer Schleife erledigen, aber auch das ist mühsam. Statt dessen werden wir die Bibliothek `fmt` verwenden, die in der nächsten Version des C++-Standards auch enthalten sein wird.

  Der Einfachheit halber gehen wir *vorerst* folgendermaßen vor, dass wir das Archiv an eine geeignete Stelle in unserem Dateisystem entpacken und in der Datei `meson.build` die `include_directories` Funktion anpassen. Das machen wir so, dass das übergebene Argument jetzt zu einer Liste (wie in Python) umgeändert wird und dort der Pfad zum `include` Verzeichnis von `fmt` hinzugefügt wird.

  Danach sind nur mehr die benötigten Headerdateien zu inkludieren. Das sind `fmt/format.h` und für die Ausgabe der `vector`-Instanz die Datei `fmt/ranges.h`.

  Da wir allerdings jede Warnung als Fehler betrachten und die Bibliothek `fmt` eine Warnung liefert, die mit `-Wpedantic` gemeldet wird, muss genau für diese Bibliothek diese Warnung deaktiviert werden:

  ```
  #pragma GCC diagnostic push
  #pragma GCC diagnostic ignored "-Wpedantic"
  #define FMT_HEADER_ONLY
  #include "fmt/format.h"
  #include "fmt/ranges.h"
  #pragma GCC diagnostic pop
  ```

  Wie du siehst, handelt es sich hierbei wieder um eine `#pragma` Direktive, die vom `gcc` (der Gnu Compiler Collection, die auch den `g++` umfasst) verstanden wird. Die Funktionsweise sollte selbsterklärend sein…

- Für die Verarbeitung der Kommandozeilenargumente werden wir hier eine *option group* von `CLI11` verwenden. Schlage dies in der Dokumentation nach!

  Die Eingabe einer variablen Anzahl an Zahlen ist ganz einfach, da dies von `CLI11` selber übernommen wird, wenn wir für die Variable einfach den Datentyp `vector<int>` verwenden.

- Es ist ein Test in der Datei `meson.build` zu erstellen (siehe `meson_tutorial`!), der für jeden Sortieralgorithmus einen TEST_CASE mittels `doctest` enthält. Strukturiere deinen Test wie im Tutorial angegeben. D.h. den Tests sei ein eigenes Verzeichnis `tests` gegönnt.

  ```
  $ meson test --list
  test1
  $ meson test test1
  ninja: Entering directory `/home/.../03_sorting/build'
  ninja: no work to do.
  1/1 test1                                  OK       0.00 s

  Ok:                   1
  Expected Fail:        0
  Fail:                 0
  ```

```
Unexpected Pass:       0
Skipped:               0
Timeout:               0

Full log written to /home/.../03_sorting/build/meson-logs/testlog.txt
```

Es kann das Testprogramm natürlich auch einfach per Hand gestartet werden:

```
$ test1
[doctest] doctest version is "2.3.3"
[doctest] run with "--help" for options
===============================================================================
[doctest] test cases:      3 |      3 passed |      0 failed |      0 skipped
[doctest] assertions:      3 |      3 passed |      0 failed |
[doctest] Status: SUCCESS!
```

4 Übungszweck dieses Beispiels

- `CLI11`
- Verwendung von `fmt`
- Implementierung der Sortieralgorithmen Bubblesort, Insertion-Sort und Selection-Sort
- `vector::size()` und `vector::data()`
- Erstellen eines einfachen Unit-Test mittels `doctest`
- `#pragma GCC diagnostic` Direktive