

CWR 2021

Vorschlag für Ihre Projektstruktur

Hinweis: Das Erstellen von makefiles und die Verwendung mehrerer Quelldateien wird ab **Zettel 2** und **3** erläutert. Dieses Dokument ist „Zusatzmaterial“. Sie müssen den Inhalt nicht beherzigen, wir glauben nur, dass er Ihnen die Arbeit erleichtern kann.

Ein wichtiges Merkmal dieses CWR-Kurses ist die Erstellung Ihrer eigenen Numerik-Bibliothek `my_numerics`. Wir wollen Ihnen so einen Einblick darin geben, wie Sie größere Projekte so strukturieren können, dass Sie Ihre Arbeit wiederverwenden und auch leicht warten können.

Wir hoffen dass Sie dadurch Ihre Arbeit dieses Semesters nicht irgendwann achtlos wegwerfen müssen, weil sie über ein unüberschaubares Durcheinander von verschiedensten Quelldateien verteilt ist, sondern dass Sie zu einem späteren Zeitpunkt auch wieder einen Blick in Ihren Code werfen können und schnell finden, wonach Sie suchen.

Wir schlagen Ihnen hier eine mögliche Ordner- und Arbeitsstruktur vor, die Sie für Ihre Zwecke adaptieren können. Sie ermöglicht Ihnen einen relativen unkomplizierten Workflow, ist aber in erster Linie für *Einsteiger* gedacht. Für „ernsthafte“/größere Projekte werden Sie einige Dinge effizienter gestalten wollen.

Beispiel

Zunächst ein Überblick über Ihren Projektordner, wie er nach einigen Wochen aussehen könnte:

CWR

 my_numerics.h

 my_numerics.h

01_Integration

 integrate.c

02_Modularisieren

 integrate.c

03_Fehlerfunktion

 errf.c

04_Gleitkommazahlen

 floats.c

05_Code-Wiederverwendung

 tests.c

 makefile

...

06_Ableitungen

 derivative.c


 derivative.o

 my_numerics.o

 derivative

 differences.csv

 makefile

 plots.py

Erklärung

1. Alle Ihre Arbeiten dieses Kurses befinden sich in einem Ordner „CWR“, dem Top-Level-Ordner (**TLO**). Dieser Ordner ist nach dem Befolgen des Tipps von Zettel 1 ein git-Repository. Wenn Sie versteckte Dateien anzeigen (`ls -a`), sollten Sie deshalb noch den Ordner `.git` und gegebenenfalls (Tipp von Zettel 2) die Datei `.gitignore` sehen.
2. Innerhalb des TLO befindet sich für jede Aufgabe ein **Aufgaben-Ordner**, bestehend aus der Aufgabennummer (damit Ihnen die Ordner sortiert angezeigt werden), sowie einem deskriptiven Namen um ihn leichter zu identifizieren.
3. Der Quellcode `my_numerics.c` und der Header `my_numerics.h` Ihrer **Numerik-Bibliothek** befinden sich im TLO. Wenn Sie etwas Ihrer Bibliothek hinzufügen wollen, dann passen Sie diese beiden Dateien an. Ihre Änderungen sind dann für sämtliche Aufgaben verfügbar.
4. In jedem Aufgabenordner befinden sich (Beispiel Aufgabe 6):
 - **Quellcode der Aufgabe**, z.B. `derivative.c`
Insbesondere ist hier die `main()`-Funktion der jeweiligen Aufgabe enthalten.
 - Die **Objects** aus der Aufgabe und von `my_numerics`.
 - Ausgabedateien, wie z.B. **CSV**-Dateien.
 - Python-Skripte und **Plots**, die Sie für die Aufgabe erstellt haben.
 - Ein **makefile**, das die Aufgabe und die Numerik-Bibliothek kompiliert und verlinkt.
5. **Wie benutze ich meine Bibliothek `my_numerics` in meinen Quellen?**
Wenn Sie Funktionen aus Ihrer Numerik-Bibliothek in einem Quellcode verwenden wollen, dann binden Sie den Header `my_numerics.h` Ihrer Bibliothek wie gewohnt am Kopf Ihrer C-Datei ein. Ihre Datei sieht dann oben vielleicht so aus:

```
#include <stdio.h>
#include <stdlib.h>
#include <tgmath.h>
#include "my_numerics.h"
```

Benutzen Sie dann ganz normal die Funktionen Ihrer Bibliothek.

6. Wie kompiliere ich meine Quellen?

Wenn Sie eine C-Quelle kompilieren wollen, die `my_numerics.h` einbindet, dann muss dem Compiler mitgeteilt werden, wo dieser Header zu finden ist. Sie können `gcc` weitere Include-Pfade mit dem Parameter `-I` mitteilen. Das könnte so aussehen:

```
gcc -I/usr/include -c source.c
```

Dieser Befehl kompiliert (-c) die Quelle `source.c` und sucht dabei zusätzlich zu den Standardpfaden im Ordner `/usr/include` nach Headern.

Um also aus Aufgabe 6 die Datei `derivative.c` zu kompilieren, geben Sie `gcc` den übergeordneten Ordner (`..`) als Suchpfad mit:

```
gcc -I.. -c derivative.c
```

7. Und meine Bibliothek?

Ihre Bibliothek kompilieren Sie der Einfachheit halber als Objekt direkt in Ihrem Aufgabenordner. Geben Sie im Terminal innerhalb Ihres Aufgabenordners folgenden Befehl ein:

```
gcc -c -I.. ../my_numerics.c
```

Das Object wird dann im Aufgaben-Ordner erstellt. Dieser Schritt ist natürlich nicht besonders effizient, denn für jede Aufgabe wird die Bibliothek neu kompiliert. Aber wir wollen es der Einfachheit halber hierbei belassen, um den Prozess so simpel wie möglich zu halten. Mit mehr Erfahrung werden Sie intelligentere Build-Prozesse entwickeln können.

Sie können die Objects anschließend in Ihrem Aufgaben-Ordner normal verlinken.

8. Und wie sieht jetzt das makefile dazu aus?

Ein Target für jeweils die beiden Objects (Aufgabencode und Bibliothek) und ein Target für die finale Binary:

```
derivative: my_numerics.o derivative.o
    gcc -I.. -Wall my_numerics.o derivative.o -o derivative
```

```
my_numerics.o: ../my_numerics.c ../my_numerics.h
    gcc -c -I.. -Wall ../my_numerics.c
```

```
derivative.o: my_numerics.o derivative.c
    gcc -c -I.. -Wall derivative.c
```