Eine bestehende Vektorimplementierung in C++ soll ergänzt werden. Diese Implementierung wird später wiederverwendet.

1 Vorbereitung

- Erstellen Sie ein Verzeichnis innerhalb Ihres Homeverzeichnis, in dem Sie alle Ihre Lösungen für das Labor programmieren werden.
- Laden Sie sich aus Ilias die Quelltexte `math.zip` hoch und kopieren die Datei in das Verzeichnis z.B. mit einem Dateibrowser.
- Öffnen Sie ein Terminalfenster (Shell) und navigieren zum erstellten Verzeichnis (`cd pfad/zu/meinem/Verzeichnis`).
- Entpacken Sie mit `unzip math.zip` die Datei. Es wird ein Unterverzeichnis `math` erstellt, indem sich der Quelltextrahmen für die Aufgabe befindet.

2 Makefile erstellen

Für den Build-Prozess wird *make* verwendet. Dieses benötigt ein *Makefile*, welches die Abhängigkeiten zwischen den Quelltexten beschreibt. Diese sind mit *cmake* in einer Datei *CMakeLists.txt* beschrieben. Das automatisch erzeugt Makefile ist für einen menschlichen Leser uninteressant.

- Im Verzeichnis `math` führen Sie das Kommando `cmake .` aus (der Punkt gehört zum Kommando). Es wird nach einem C++-Compiler gesucht (der auch gefunden werden sollte). Neben einem neuen Unterverzeichnis werden einige Dateien inklusive dem Makefile erzeugt.
- Führen Sie `make` aus. Das bestehende Programm `math_test` sollte fehlerfrei übersetzt werden. Es werden nur Compileraufrufe für ein Ziel getätigt, wenn die abhängigen Quelltexte einen aktuelleren Zeitstempel besitzen.
- Führen Sie das erstellte Testprogramm `math_test` aus (ggf. als `./math_test`, wenn das aktuelle Verzeichnis nicht im Suchpfad angegeben ist). Es dürfen keine Tests fehlschlagen.
- Sie können alle vom Compiler erzeugen Dateien mit `make clean` löschen. Dies ist manchmal sinnvoll, da manche Abhängigkeiten z.B. vorübersetzte Headerdateien nicht von cmake oder make erkannt werden.
- Führen Sie `cmake --help` aus. Am Ende werden alle verfügbaren Generatoren angegeben, um für unterschiedliche Plattformen Makefiles zu produzieren.

3 Quelltext ergänzen

- Im Header `math.h` sind zwei Objektmethoden, welche die Länge und das Quadrat der Länge eines Vektors zurückgeben, und eine Templatefunktion, die das Skalarprodukt zweier Vektoren berechnet, auskommentiert.
- In der Implementierungsdatei `math.tcc` fehlen die zugehörigen Implementierungen. Alle anderen Implementierungen, die auf diese fehlenden Funktionalitäten basieren sind auskommentiert.

- In `math.cc` ist die Instantiierung des fehlenden Skalarprodukts auskommentiert.
- In `math_test.cc` sind alle Tests für die auskommentierten Funktionen ebenfalls auskommentiert.

1. Implementieren Sie schrittweise die drei fehlenden Funktionen. Testen Sie sie mit eigenen Tests, die sie in `math_test.cc` hinzufügen. Es wird Google-Test (gtest) verwendet. Verlassen Sie sich nicht auf die bestehenden Tests.
2. Zum Schluss aktivieren Sie die vorhanden Tests und Implementierungen, indem Sie die Kommentare darum herum entfernen. Die Tests müssen alle funktionieren.

Weitere Informationen zu `gtest` erhalten Sie unter `https://google.github.io/googletest`.

Hinweise zur CMakeLists-Datei

Die Datei `CMakeLists.txt` ist sehr einfach aufgebaut:

```
cmake_minimum_required (VERSION 3.9)
project (math)

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED True)

add_compile_options(-g -Wall -Wextra -Wpedantic -Wl,--stack,16777216)

add_executable(math_test math_test.cc math.cc)
target_link_libraries(math_test gtest gtest_main)
```

- Die erste Zeile definiert die Mindestversion des benötigten cmake.
- In der zweite Zeile wird der Name des Projekts definiert. Dies ist in unseren Beispiel eigentlich unnötig.
- Mit `set` können verschiedene cmake-Variablen, aber auch Betriebssystem-Umgebungsvariablen, gesetzt werden. Im Beispiel wird mindestens ein C++-20 Standard zwingend gefordert.
- Mit `add_compile_options` werden Compileroptionen des verwendeten (von cmake gefundenen) Compilers angegeben. Die Optionen sind Compiler-spezifisch. Es werden alle Warnungen angeschaltet und der call stack vergrößert.
- Mit `add_executable` wird für eine vom Compiler zu erzeugende ausführbare Datei `math_test` die zugehörigen benötigten Quelltextdateien angegeben. Im Beispiel ist das die GoogleTest-Datei `math_test.cc` (im Normalfall wäre hier ein Quelltext mit der `main`-Funktion angegeben). Die Testdatei hängt von der C++-Implementierung `math.cc` ab für welches die Objektdatei vom Compiler erzeugt wird. Die Abhängigkeit der Datei `math.cc` von der Header- und Templatedatei wird (hier unter Unix) automatisch des von cmake verwendeten Makefile-Generators erkannt. Wenn Sie mit `touch math.h` den Zeitstempel der Headerdatei aktualisieren, sollte bei Aufruf von `make` wieder alles übersetzt werden. Analog bei der `.tcc`-Datei.

- In der letzten Zeile wird für den Linker definiert, welche zusätzlichen (nicht-Standard) Bibliotheken benötigt werden, um die ausführbare Datei zu erzeugen. `gtest` enthält die Funktionen des GoogleTest-Frameworks, `gtest_main` enthält die für eine ausführbare Datei benötigte `main`-Funktion.

Weitere Informationen zu cmake finden Sie unter `https://cmake.org`.

Hinweise für MinGW unter Windows

Sie müssen neben C++-Compiler der GNU-Compiler-Collection auf jeden Fall cmake, make und gtest für MinGW installieren (normalerweise über den MinGW Paketmanager `pacman`)

- Das Makefile sollte mit `cmake -G "MinGW Makefiles" .` erzeugt werden, da ansonsten vermutlich der Ninja-Makefile-Generator verwendet wird.
- Das make-Kommando bei MinGW heißt `mingw32-make` und nicht `make`.

Hinweise für Macs

Auf jeden Fall sollten Sie prüfen, ob tatsächlich `g++` der GCC-Compiler ist, da sich oft dahinter der `clang`-Compiler verbirgt und `g++` nur ein Alias ist.

Bei Aufruf von `g++ --version` muss im ausgegebene Text der Hinweis auf GCC vorkommen und nicht clang (diese Aufgabe funktioniert aber sicherlich auch noch mit clang, spätere aber vielleicht nicht mehr).