

Einführung in die Programmierung mit C++

Übungsblatt 12

Klassenhierarchien, Dynamische Speicherverwaltung, und Make

Sebastian Christodoulou, Alexander Fleming, and Uwe Naumann

Informatik 12:

Software and Tools for Computational Engineering (STCE)

RWTH Aachen

In dieser Übung soll eine (kleine) Lineare Algebra Bibliothek implementiert und gebaut werden. Diese Bibliothek enthält die Klassen *Matrix* und *Vector*, die Matrixen in $\mathbb{R}^{n \times n}$ und Vektoren in \mathbb{R}^n darstellen. Die zusätzliche Klassen *VectorView* und *ConstVectorView* stellen auch Vektoren in \mathbb{R}^n dar und erlauben es, dynamische Felder von Typ `double` zuzugreifen ohne dass das Objekt des *View*-Typs ein dynamisches Feld besitzen muss.

Einige Funktionen, die Vektor-Vektor, Matrix-Matrix, und Matrix-Vektor Operationen durchführen, sind auch außerhalb Klassen deklariert. Diese sollen auch implementiert werden!

Um die Dokumentation dieser Bibliothek zu sehen, kann man `make doc` ausführen und `index.html` aufmachen.

► *TrackedViewable*

- Etwas, von dem "Views" genommen werden können.
- Die Funktionen `increment_live_views()` und `decrement_live_views()` werden immer aufgerufen wenn ein *ConstVectorView* oder *VectorView* konstruiert bzw. dekonstruiert wird.
- Diese Basisklasse ist schon implementiert.

► *abstrakt VectorLike*

- Stellt ein Vektor in \mathbb{R}^n dar und erlaubt Lesezugriff auf seine Einträge.
- Deklariert die funktionen `virtual int size()const`,
`virtual double at(int i)const`, und `bool check_bounds(int i)const`.
- Der Destruktor wird hier als *virtual* deklariert! (Warum, eigentlich?)

► *abstrakt MutableVectorLike*: *public VectorLike*

- Erweitert die *VectorLike* Schnittstelle mit Lesezugriff und Zuweisungsoperationen.
- Deklariert `virtual double& at(int i)`.
- Deklariert einige Zuweisungsoperationen.

► *ConstVectorView*: *public VectorLike*

- ▶ Etwas, das als ein Vektor in \mathbb{R}^n betrachtet wird und ein Teil eines *VectorLikes* oder einer *Matrix* ist.
- ▶ Deklariert `ConstVectorView cview(int start, int length) const`.
- ▶ *VectorView*: `public MutableVectorLike`
 - ▶ Etwas, das als ein Vektor in \mathbb{R}^n betrachtet wird und ein Teil eines *MutableVectorLikes* oder einer *Matrix* ist.
 - ▶ Deklariert `ConstVectorView cview(int start, int length) const` und `VectorView view(int start, int length)`.
 - ▶ Diese Klasse erbt von *MutableVectorLike* und hat deswegen Schreibzugriff auf den unterliegenden Speicher.
- ▶ *Vector*: `public MutableVectorLike, public TrackedViewable`
 - ▶ Ein Vektor in \mathbb{R}^n .
 - ▶ Besitzt und verwaltet Speicher.
- ▶ *Matrix*: `public TrackedViewable`
 - ▶ Eine Matrix in $\mathbb{R}^{n \times n}$.
 - ▶ Besitzt und verwaltet Speicher.
 - ▶ Ihre Einträge werden in "Row-Major" Ordnung gespeichert. (Siehe [Übung 5](#)).

📁 skeleton: Hauptordner.

📄 Makefile

📄 main.cpp: Das Testprogramm

📁 input: Inputdateien für das Testprogramm.

📁 doc: Doxygen-Dokumentation der Bibliothek.

📁 include:


📄 linalg.hpp: Headerdatei der Bibliothek.

📁 library: Quellcodedaten für die Bibliothek.

📄 Makefile: Sub-Makefile, die die statische Bibliothek baut.

1. Führe `make doc` aus, um die Dokumentation der Bibliothek anzusehen.
2. Schreibe Quellcode Dateien, die alles in der Bibliothek, was noch nicht implementiert ist, implementieren.
3. Ergänze die Makefiles `skeleton/Makefile` und `skeleton/library/Makefile` damit `make main.exe` die Bibliothek `liblinalg.a` baut und das Testprogramm kompiliert.
4. Prüfe, ob das Testprogramm den korrekten Output ergibt. Dieser ist in der Dokumentation genau beschrieben.

Abgabe:

-  `skeleton.zip`: Einfach `skeleton` wieder archivieren und hochladen! Man soll nach Abgabe mit `make all && ./main.exe input/input.txt` das Testprogramm ausführen können.