Mathematisches Praktikum — Sommersemester 2018
Prof. Dr. Arnold Reusken — Sven Gross, Hauke Sass, Johannes Stemick

igpm Institut für Geometrie und Praktische Mathematik | RWTH AACHEN UNIVERSITY

Aufgabe 2, Teil II

Bearbeitungszeit: eine Woche (bis Montag, 7. Mai 2018)

Mathematischer Hintergrund: einfache Sortieralgorithmen

Elemente von C++: Template-Funktionen, Strukturen, Operatorüberladung, Makefiles

Aufgabenstellung

In der letzten Aufgabe haben Sie verschiedene Sortierfunktionen für Felder vom Typ `unsigned int` programmiert. Schreiben Sie die Funktionen, die Bubblesort, das Auswahlsortieren und Mergesort realisieren, so um, dass Felder eines beliebigen Datentyps `ElemT` sortiert werden können, für den ein Vergleichsoperator `<` zur Verfügung steht[1]. Die Felder sollen dabei über `std::vector<ElemT>` realisiert werden, einen der Container-Datentypen der C++ Standard Library[2][3]. Der Template-Parameter `ElemT` steht dabei stellvertretend für einen Datentyp, der den Typ der Einträge des Feldes festlegt und erst zur Compile-Zeit spezifiziert werden muss.

Einlesen der Felder aus Dateien

In dieser Aufgabe sollen insgesamt drei verschiedene Felder sortiert werden, die aus zur Verfügung gestellten Dateien eingelesen werden sollen: ein `double`-Feld (`doubles.txt`), ein `string`-Feld (`strings.txt`) sowie ein `Student`-Feld[4] (`studenten.txt`). Schreiben Sie dazu eine Template-Funktion

```
template<typename ElemT>
void einlesen( std::ifstream& ifs, std::vector<ElemT>& feld)
```

mit Template-Parameter `ElemT`, die aus dem *input file stream* `ifs` die entsprechenden Daten liest und im Vektor `feld` ablegt. Dabei können Sie mithilfe von `!ifs.fail()` bzw. `ifs.eof()` abfragen, ob die letzte Leseoperation mit dem Eingabeoperator `>>` erfolgreich war bzw. ob das Ende der Datei (*end of file*) erreicht wurde. Da die Anzahl der zu lesenden Daten vorher nicht feststeht, empfiehlt es sich, die Einträge mit Hilfe von `push_back` an den Vektor `feld` anzuhängen.

Ausgabe der Felder

Zur Ausgabe der Felder schreiben Sie analog eine Template-Funktion

```
template<typename ElemT>
void ausgeben( std::ostream& os, const std::vector<ElemT>& feld)
```

die einen Vektor `feld` in den *output stream* `os` ausgibt. `os` kann beim Aufruf der Funktion z.B. `std::cout`, aber auch ein *output file stream* vom Typ `std::ofstream` sein.

[1] Dieser Vergleichsoperator muss dazu natürlich eine strikte schwache Ordnung definieren.
[2] Manchmal auch als STL bezeichnet. Streng genommen ist die STL aber ein Vorläufer der C++ Standard Library.
[3] Eine Dokumentation der C++ Standard Librarz finden Sie z.B. unter [2] oder [1].
[4] Der Datentyp `Student` ist weiter unten beschrieben.

Der benutzerdefinierte Datentyp `Student`

In den Dateien `student.h` und `student.cpp` stellen wir Ihnen bereits ein Grundgerüst für den neuen Datentyp `Student` zur Verfügung. Ihre Aufgabe besteht darin, den Ausgabeoperator `<<` für Variablen vom Typ `Student` zu ergänzen. Außerdem sollen Sie die Vergleichsoperatoren `<`, `==` und `!=` für den Datentyp `Student` definieren. Die Sortierung der Studenten soll dabei aufsteigend nach Nachnamen erfolgen, bei gleichem Nachnamen nach Vornamen. Falls Sie in ihrem Sortieralgorithmus auch `>` benutzen, so müssen Sie natürlich auch diesen Operator zur Verfügung stellen.

In der Header-Datei `student.h`, welche die Schnittstelle zur Benutzung des neuen Datentyps `Student` darstellt, sollen nur Deklarationen (z.B. der Operatorfunktionen) auftauchen. Die eigentlichen Definitionen der Funktionen gehören in die Datei `student.cpp`.

Hauptprogramm

Gestalten Sie Ihr Hauptprogramm so, dass zunächst die Daten aus den Dateien `doubles.txt`, `strings.txt` und `studenten.txt` eingelesen werden (`fstream` einbinden!). Anschliessend soll der Benutzer angeben, welches der drei Sortierverfahren (B=Bubblesort, A=Auswahlsortieren, M=Mergesort) angewandt werden soll. Nach der Sortierung sollen die Felder jeweils auf dem Bildschirm ausgegeben werden und zur Überprüfung an die Funktion `ergebnis` der Praktikumsumgebung übergeben werden.

Makefiles

In dieser Aufgabe ist Ihr Programmcode zum ersten Mal auf mehrere Dateien verteilt. Das hat zur Folge, dass bei Verändern einer dieser Dateien verschiedene Objektdateien neu übersetzt werden müssen und die ausführbare Datei neu gelinkt werden muss. Um dabei nicht den Überblick zu verlieren und das Verfahren zu automatisieren, kann man z.B. sogenannte *Makefiles* verwenden.

Makefiles bestehen aus sogenannten Regeln, die folgende Form aufweisen:

```
Ziel: Abhängigkeiten
<Tab> Befehl
```

<Tab> steht dabei für das Tabulatorzeichen, was an dieser Stelle zwingend erforderlich ist, um die Befehlszeile zu markieren. Die Regel

```
myProg.o: myProg.cpp student.h
<Tab> g++ -c myProg.cpp
```

bedeutet z.B., dass die Objektdatei `myProg.o` immer dann neu erzeugt werden muss, wenn sich eine der Dateien `myProg.cpp` oder `student.h` verändert hat. Eine umfassende Dokumentation finden Sie unter [3].

Für diese Aufgabe stellen wir Ihnen ein `Makefile` zur Verfügung, das Sie evtl. noch auf Ihre Bedürfnisse hin anpassen müssen. Ein Aufruf von `make` auf der Kommandozeile führt die notwendigen Regeln aus und ersetzt damit die sonst nötigen Compiler-Aufrufe.

Literatur

[1] *C++-Dokumentation.* `http://www.cplusplus.com`.

[2] *C/C++-Referenz.* `http://www.cppreference.com/index.html`.

[3] *Dokumentation zu GNU Make.* `http://www.gnu.org/software/make/`.