**03 Notenstatistik**

**Zusammenfassung der Problemstellung**

Das Programm soll eine Anzahl von Punkten von Studierenden und die benötigte Punktzahl für die Note 6 einlesen. Im Anschluss sollen die Noten der einzelnen Studierenden berechnet werden. Die Ausgabe am Schluss ist eine Statistik über das gesamte und enthält Anzahl vorkommen jeder Note, Angaben wie beste- und schlechteste vorkommende Note, Notendurchschnitt, Anzahl Studierende und Anzahl Studierende, die bestanden haben (Note > =4).

**Lösungsvarianten**

Um die Berechnung aller benötigten Informationen beieinander zu haben gibt es ein struct, das alle Variablen enthält. Es gibt eine Funktion, welche die Noten aus den Punkten berechnet, eine Funktion, welche die Werte für die Statistik berechnet und eine Funktion für die Ausgabe der Statistik. Die main-Funktion liest die Daten ein und übergibt sie den Funktionen. Wir haben uns für zusätzliche Funktionen entschieden, um verschiedene Aufgaben innerhalb des Programms klar aufzuteilen und damit Übersicht zu schaffen.

**Modulübersicht**

Da es sich um ein sehr kleines Programm handelt, befindet sich der ganze Code in main.c, aufgeteilt auf die 4 Funktionen main, get\_statistics, get\_mark, print\_statistics.

**Globale Variablen:**

* struct Statistics

**Lokale Variablen:**

* int pointlist[100] (main)
* int len (main)
* int all\_points\_inserted (main)
* int points\_6 (main)
* Statistic statistics (main)
* Char rerun (main)
* int grades[len]
* int mark6 (get\_statistics)
* int mark5 (get\_statistics)
* int mark4 (get\_statistics)
* int mark3 (get\_statistics)
* int mark2 (get\_statistics)
* int mark1 (get\_statistics)
* int best\_mark (get\_statistics)
* int worst\_mark (get\_statistics)
* double average\_mark (get\_statistics)
* int i (get\_statistics)
* double mark (get\_mark)

**Programmablauf**

Main > ruft get\_statistics auf, zur Berechnung aller für die Ausgabe benötigten Infos. Get\_statistics ruft innerhalb noch get\_mark auf, wo aus den Punkte die Note berechnet und zurückgegeben wird. Get\_statistics gibt das struct Statistics zurück an die main-Funktion, die zum Schluss print\_statistics aufruft alle Angaben auf der Konsole ausgibt.

**Tests**

*test\_berechnung\_schaltjahr:* prüft die Berechnung des Folgetages im spezialfall Schaltjahr

*test\_berechnung\_monatwechsel:* prüft die Berechnung des Folgetages für einen normalen Monatswechsel

*test\_berechnung\_jahreswechsel:* prüft die Berechnung des Folgetages für einen normalen Jahreswechsel

*test\_berechnung\_erstes\_datum:* prüft die Berechnung des Folgetages für das erste gültige Datum

*test\_teste\_gueltigkeit\_vor\_erstem\_datum:* prüft, ob der Gültigkeitstest ein Datum vor dem ersten gültigen Datum ablehnt

*test\_teste\_gueltigkeit\_von\_datum\_tag:* prüft, ob der Gültigkeitstest ein ungültiges Datum ablehnt

*test\_teste\_gueltigkeit\_von\_datum\_monat:* prüft, ob der Gültigkeitstest ein ungültiges Datum ablehnt

*test\_teste\_gueltigkeit\_von\_datum\_in\_zukunft:* prüft, ob der Gültigkeitstest ein Datum in der Zukunft akzeptiert

Alle Tests liefen erfolgreich durch.

**Erkenntnisse**

Das Programm läuft fehlerfrei.

Die Funktion get\_statistics könnte noch weiter auf mehrere Funktionen aufgeteilt werden. Z.B das zählen, wie oft jede Note vorkommt, könnte ausgelagert werden. Eine weitere Möglichkeit einer Umstrukturierung wäre stattdessen get\_mark ebenfalls von der main-funktion aus aufzurufen. Jedoch macht es entsprechend der Aufteilung nach Zuständigkeiten, mehr Sinn, wenn get\_statistics sich darum kümmert.

Mögliche Erweiterungen

**Anhang A: Quelltext**