PROGC Praktikum 3 – Arrays

Dauer: 2 Lektionen

Aufgabe: Notenstatistik

Schreiben Sie ein C-Programm, welches es einem Dozenten ermöglicht, die Notenstatistik zu einer Prüfung zu erstellen. In einem ersten Schritt sollen dabei die Punktzahlen, die die einzelnen Studenten erzielt haben, eingegeben und in einem Array gespeichert werden. Wählen Sie dazu den Array genügend gross (z.B. 100 Elemente). Ist diese Eingabe abgeschlossen, wird die minimale Punktzahl eingegeben, welche für die Note 6 notwendig ist. Danach berechnet das Programm die Notenstatistik, welche in etwa folgendermassen aussehen soll:

```
-------------------------------------------------------
Statistics (20 students, 19 points needed for mark 6):

  Mark 6: 2
  Mark 5: 7
  Mark 4: 7
  Mark 3: 4
  Mark 2: 0
  Mark 1: 0

  Best mark:    6
  Worst mark:   3
  Average mark: 4.35
  Mark >= 4:    16 students (80.00 %)
-------------------------------------------------------
```

Ist diese Statistik ausgegeben worden, soll der Benutzer jeweils gefragt werden, ob er das Programm beenden oder eine neue Notenstatistik für die eine andere minimale Punktzahl für die Note 6 eingeben will. Im zweiten Fall wird der Benutzer nach dieser minimalen Punktzahl gefragt und das Programm gibt die neue Notenstatistik aus.

Hinweise:

- Geben Sie die erreichten Punkte der Studenten einzeln ein, so dass das Programm immer eine Punktzahl einliest und dann nach der nächsten fragt. Nehmen Sie an, dass Punkte immer ganzzahlig und ≥ 0 sind. Die Eingabe einer Punktzahl " –1" soll die Eingabe der Punkte komplettieren; alle anderen negativen Punktzahlen sollen in einer Fehlermeldung resultieren.
- Noten seien ebenfalls ganzzahlig; 6 ist die beste und 1 die schlechteste Note. Ist *p6* die Anzahl Punkte, die für eine 6 nötig sind und *p* die Anzahl Punkte, die ein Student erreicht hat, dann wird seine Note *n* wie folgt berechnet: *n = (5 * p / p6) + 1*. Noten werden auf eine ganzzahlige Note gerundet, wobei ".5" oder besser aufgerundet wird (4.4 gibt zB Note 4; 4.5 gibt Note 5).
- Zum Auf- oder Abrunden gibt es zwei Funktionen in der C Standard Library, die Sie bei Bedarf verwenden können. Die Funktionen werden mit #include <math.h> eingebunden:

  `double ceil(double n)` gibt den kleinsten ganzzahligen Wert ≥ n zurück
  `double floor(double n)` gibt den grössten ganzzahligen Wert ≤ n zurück

  Linux-Benutzer müssen dabei beim Kompilieren evtl. die math-Library explizit spezifizieren, weil die math-Funktionen oft nicht in der üblichen Standard Library (libc.a) integriert sind, sondern sich in einer separaten Library (libm.a) befinden:

```
gcc –lm programm.c
```

- Achten Sie auf eine übersichtliche Programmstruktur und verwenden Sie Funktionen, wo dies sinnvoll ist. Mit zB funktionsname(int a[], int len) können Sie einen int-Array a einer Funktion als Parameter übergeben. Weil ein Array seine Länge nicht kennt, ist es oft zweckmässig, zu jedem Array einen weiteren Parameter (hier len) zu übergeben, der dessen Länge spezifiziert.
- Wenn Sie Mühe haben, eine eigene Programmstruktur zu finden, ist hier eine Möglichkeit angegeben. Natürlich gibt es eine Vielzahl von anderen, guten Möglichkeiten.
    - In der main-Funktion werden zuerst die Punktzahlen eingegeben, anschliessend die nötige Punktzahl für eine Note 6.
    - Die Notenliste und die Punktzahl für eine 6 werden einer Funktion getStatistics übergeben, welche sämtliche Werte für die auszugebende Statistik berechnet. Diese Werte sollen mittels einer geeigneten Struktur von der Funktion zurückgegeben werden. Die Struktur selbst soll unter anderem einen Array enthalten, der angibt, wie viel mal jede Note vorgekommt.
    - Das Berechnen der Note aus einer Punktzahl und der nötigen Punktzahl für eine 6 soll in einer Funktion getMark durchgeführt werden. Diese Funktion wird von getStatistics verwendet.

Aufgabe: Unit Tests für Notenstatistik

Bestimmte Teile der Applikation sollen nun mit einem Komponententest (Unit Test) getestet werden. Dieser überprüft nicht wie die Tests in Praktikum 1 und 2 das Programm als Ganzes, sondern gewisse Module davon. Es sollen die Funktionen getStatistics und getMark getestet werden.

Verfahren von Unit Tests:

1. Ausgangszustand initialisieren.
2. Ausführen der zu testenden Operation.
3. Vergleichen der Resultate mit Sollwerten.

Normalerweise werden die zu testenden Funktionen in das Testprogramm / Testframework eingebunden. Da zu diesem Zeitpunkt die Modulare Programmierung im Unterricht noch nicht behandelt wurde, sollen die Tests in derselben C-Datei programmiert werden. Mit den Präprozessor-Direktiven #define, ifdef, #ifndef, #if, #elif, #else und #endif soll zwischen normaler Programmausführung und Testmodus gewechselt werden können.

Schreiben Sie Ihr Programm so, dass zwischen dem normalen Modus und einem Testmodus gewechselt werden kann. Sie können dazu das untenstehende Grundgerüst verwenden. Falls eine Variable TESTING definiert wurde, wird die main-Funktion des Testprogrammes ausgeführt. Andernfalls (bei auskommentiertem #define) wird die main-Funktion des normalen Programmablaufs verwendet.

```
// #define      TESTING

#ifdef TESTING
// Testprogramm
main(…)  {…}

#else /* TESTING not defined */
// Normales Programm
main(…) {…}

#endif
```

```
#define      TESTMODE 1

#if TESTMODE==1
// Testprogramm
main(…)  {…}

#elif TESTMODE==0
// Normales Programm
main(…) {…}

#endif
```

Schreiben Sie je einen Test für die Funktionen getStatistics und getMark folgendermassen:

- Die Ausgangszustände initialisieren. Die Punktzahlen und Noten werden nicht mehr über die Tastatur eingegeben, sondern direkt in die Variablen und Arrays geschrieben. Da die Eingaben fix sind, sind die Resultate zu erwartenden, die die Funktion ausgeben muss.
- Die Funktion aufrufen.
- Die Resultate auf ihre Richtigkeit prüfen. Binden Sie dazu die Header-Datei assert.h mittels **#include <assert.h>** in den Quellcode ein. Dadurch steht Ihnen die Funktion **void assert (int expression)** zur Verfügung. Als Parameter wird ein Ausdruck verlangt (z.B. „minPoints6 == 25“ oder „queue1->iter == NULL“). Ist dieser Ausdruck wahr, so fährt der Test fort. Andererseits wird auf sdterr eine Fehlermeldung ausgegeben. Der Test wird bei einem Fehlschlag über abort abgebrochen. Wählen Sie als Parameter sinnvolle Ausdrücke, welche die Funktion gründlich testen.