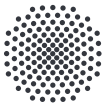
Universität Stuttgart
Institut für
Informationssicherheit

InformatikII - Programmierung in C++
Sommersemester 2023
Maria Unger-Zimmermann

Übungsblatt 5

15.05, 16.05. und 19.05

Problem 5.1: Vektoren, Teil 2: Primzahlliste

Nun eine Anwendung von Vektoren: Wir werden alle Primzahlen kleiner als 100 bestimmen – dabei nutzen wir aus, dass eine Zahl $8 \leq x < 100$ genau dann Primzahl ist, wenn sie durch keine der vier kleinsten Primzahlen 2, 3, 5 oder 7 teilbar ist (denn 11^2 ist ja schon 121); zur Kontrolle: es gibt genau 25 Primzahlen kleiner als 100, nämlich

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97$$

1. Schreiben Sie als Vorbereitung ein Programm, in dem Sie in einer Schleife alle Zahlen $8 \leq x < 100$ auf Teilbarkeit durch 2, 3, 5 oder 7 testen und die Primzahlen davon ausgeben (mit einem zusätzlichen Leerzeichen nach jeder Zahl liest sich des Ergebnis angenehmer…).
2. Erweitern Sie nun Ihr Programm, so dass die Primzahlen in einem Vektor gespeichert werden. Dazu sind drei Dinge hilfreich:
   - Ein Vektor kann eine Initialisierungsliste bekommen:
     ```
     vector<int> primzahlen {2, 3, 5, 7};
     ```
     erzeugt z.B. einen Vektor, der bereits die vier kleinsten Primzahlen enthält.
   - ein Vektor kann wachsen, indem wir mittels z.B. `primzahlen.push_back(i)` ein Element hinten anfügen.
   - Die aktuelle Länge (Anzahl der Elemente) des Vektors `primzahlen` bekommen wir mit `primzahlen.size()`.
3. Schreiben Sie nun eine weitere Schleife, in der Sie die Elemente der Reihe nach ausgeben. Ein Leerzeichen zwischen den Elementen tut's nach wie vor – schöner wären allerdings Kommas, wobei vor dem ersten und nach dem letzten Element keins stehen sollte, also eine Fallunterscheidung nötig wird.
4. Da Sie die Zahlen 2, 3, 5 und 7 jetzt im Vektor stehen haben, können Sie den Primzahltest auch als Schleife über die ersten vier Elemente programmieren.

   Wenn Sie das gemacht haben, ist es einfach, auch Primzahlen jenseits der 100 zu bestimmen.

Problem 5.2: Funktionen ohne Rückgabewert

Das Schreiben von Funktionen ist am Anfang etwas gewöhnungsbedürftig, daher gehen wir es vorsichtig an. In dieser Aufgabe haben wir erstmal nur Funktionen ohne Rückgabewert (also mit `void` statt Rückgabetyp). Die Wirkung des Funktionsaufrufs ist dann auch nicht die Berechnung eines Rückgabewertes, sondern ein *Seiteneffekt* – hier die Ausgabe auf dem Bildschirm (`cout`).

1. Schreiben Sie zunächst eine Funktion `drucke_dreieck` ohne Parameter:

   ```
   void drucke_dreieck() {
      // Hier kommen die Anweisungen hin
   }
   ```

   ```
      X
     XXX
    XXXXX
   XXXXXXX
   ```

   In der Funktion soll wie in Aufgabe **??**.**??** eine Variable n vom Typ `int` definiert und vom Benutzer eingegeben werden und das Dreiecksmuster gedruckt werden (im Kasten sehen Sie das Ergebnis für $n = 4$; den Programmcode für das Drucken können Sie vermutlich einfach aus Ihrer Lösung von Aufgabe **??** kopieren).

   In der Funktion `main` soll die Funktion `drucke_dreieck` aufgerufen werden.

2. Ändern Sie nun Ihr Programm so, dass die Funktion `drucke_dreieck` einen Parameter n vom Typ `int` bekommt und entfernen Sie die Definition und das Einlesen von n im Funktionsrumpf.

   In der Funktion `main` soll die Funktion `drucke_dreieck` dreimal aufgerufen werden: Je einmal mit Parameterwert 3, 5 und 9.

3. Nun noch ein komplizierteres Muster: Schreiben Sie eine Funktion `drucke_etage` mit drei Parametern `start`, `ende` und `leerzeichen` vom Typ `int`. Es soll für alle Werte i im Bereich `start`…`ende` je eine Zeile drucken:

   - Zuerst so viele Leerzeichen, wie `leerzeichen` angibt,
   - dann Zeile i von unserem Dreiecksmuster.

   Der Aufruf `drucke_etage(3, 5, 2)` würde z.B. Folgendes drucken (␣ ist ein Leerzeichen):

   ```
   ␣␣␣␣XXXXX
   ␣␣␣XXXXXXX
   ␣␣XXXXXXXXX
   ```

Schreiben Sie im `main` eine `for`-Schleife, die durch geeignete Aufrufe von `drucke_etage` folgende prachtvolle Fichte druckt:

```
                    X
                   XXX
                  XXXXX
                 XXXXXXX
                  XXXXX
                 XXXXXXX
                XXXXXXXXX
               XXXXXXXXXXX
                XXXXXXXXX
               XXXXXXXXXXX
              XXXXXXXXXXXXX
             XXXXXXXXXXXXXXX
              XXXXXXXXXXXXX
             XXXXXXXXXXXXXXX
            XXXXXXXXXXXXXXXXX
           XXXXXXXXXXXXXXXXXXX
          XXXXXXXXXXXXXXXXXXXXX
```