Fakultät IV - Elektrotechnik und Informatik
Einführung in die Programmierung WS 2017/18
Feldmann / Faucard / Lichtblau / Streibelt / Krüger / Enghardt / Palmer

Aufgabenblatt 6

letzte Aktualisierung: 17. Oktober, 15:55 Uhr

(1cb9b27e3db5adc6c788cb02cd30cef870078500)

Ausgabe: Dienstag, 24.10.2017
Abgabe: Donnerstag, 26.10.2017, 21:59

Thema: *Pointer Syntax & Speicher*

1 Abgabemodalitäten

1. Die Aufgaben des C-Kurses bauen aufeinander auf. Versuche daher bitte Deine Lösung noch am gleichen Tag zu bearbeiten und abzugeben.
2. Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des tubIT/IRB mittels `gcc -std=c99 -Wall` kompilieren.
3. Die Abgabe erfolgt ausschließlich über unser SVN im Abgaben-Ordner.
4. Du kannst bis zur Abgabefrist beliebig oft neue Versionen abgeben.
5. Die Abgabe erfolgt in folgendem Unterordner:

   `ckurs-wise1718/Studierende/<L>/<tubIT-Login>/Abgaben/Blatt0<X>`

   wobei `<L>` durch den ersten Buchstabe des TUBIT-Logins und `<X>` durch die Nummer des Aufgabenblattes zu ersetzen sind. Die Ordner werden automatisch angelegt sobald die Abgabe freigeschaltet wird.
6. Benutze für alle Abgaben soweit nicht anders angegeben das folgende Namensschema: `ckurs_blatt0<X>_aufgabe0<Y>.c` wobei `<X>` und `<Y>` entsprechend zu ersetzen sind. Gebe für jede Unteraufgabe genau eine Quellcodedatei ab.
7. Du darfst den Abgabeordner für das Blatt nicht selbst erstellen, das machen wir jeden Morgen kurz nach 8 Uhr!
8. Du musst aber den Befehl `svn up` auf der obersten Verzeichnisebene des Repositories (also in `ckurs-wise1718`) ausführen um alle Änderungen vom Server abzuholen.
9. Im Abgaben-Ordner gelten einige restriktive Regeln. Dort ist nur das Einchecken von Dateien mit den in der Aufgabe vorgegebenen Namen erlaubt, ausserdem werden die Abgabefristen vom Server überwacht. Beachte eventuelle Fehlermeldungen beim SVN-Commit. Lade nur Dateien hoch, die Du selbst bearbeiten sollst, insbesondere also keine Vorgaben.
10. Es gibt einen Ordner 'Arbeitsverzeichnis', in dem Du Dateien für Dich ablegen kannst.
11. Die Ergebnisse der automatischen Tests kannst Du auf OSIRIS einsehen:
    https://teaching.inet.tu-berlin.de/services/osiris-wise1718/

Sieb des Eratosthenes

Primzahlen zwischen 2 und n können mit dem nach *Eratosthenes von Kyrene* bezeichneten Verfahren berechnet werden.

Datenstruktur
Zu Grunde gelegt wird dem Verfahren ein Array, das für alle Zahlen z_i zwischen 2 und n angibt, ob es sich um Primzahlen (ausgedrückt als 1) oder Nicht-Primzahlen (ausgedrückt als 0) handelt. Zunächst wird davon ausgegangen, dass alle Zahlen Primzahlen sind. Im Verlauf des Programm wird das nach und nach korrigiert, indem alle Mehrfache von Primzahlen als Nicht-Primzahlen markiert werden.
Achtung: Da in `C` die Indexierung von Arrays mit 0 beginnt, laufen die Indizes von 0 bis $n-2$, während die repräsentierten Zahlen von 2 bis n laufen. Anders ausgedrückt: Das Index i des Array repräsentiert die Zahl z_i, also $z_i = i + 2$.

Verfahren
Das Verfahren ist im Folgenden Schritt für Schritt erklärt:

1. Deklariere ein Array, mit einem Element für jede Zahl zwischen 2 und inklusive n.
   (Also der Größe $n-1$)
2. Initialisiere jedes Element der Liste auf 1. (Wir nehmen an, dass es eine Primzahl ist.)
3. Für jedes i'te Element zwischen 2 und n mache folgendes:
   - Wenn das Element am Index i einen Wert von 1 (d.h. es ist eine Primzahl) besitzt:
     - Setze alle Mehrfache dieser Zahl z_i auf 0 (d.h. es ist keine Primzahl). Dies wird erreicht indem alle Mehrfache x der Indizes $i + x * z_i = i + x * (i + 2)$ auf 0 gesetzt werden. (Grund: Jedes Mehrfache von einer Primzahl ist keine Primzahl.)
4. Alle Zahlen z_i für die gilt `array[i] == 1`, sind nun tatsächlich Primzahlen.

Beispiel für $n = 9$

1. Initialisiere das Array.

   ```
   Index (i): 0, 1, 2, 3, 4, 5, 6, 7
   Zahl (z_i):  2, 3, 4, 5, 6, 7, 8, 9
   array[i]:  1, 1, 1, 1, 1, 1, 1, 1
   ```

2. Für Zahl $z_i = 2$ ($i = 0$), werden alle mehrfache von 2 auf 0 gesetzt

   ```
   Index (i): 0, 1, 2, 3, 4, 5, 6, 7
   Zahl (z_i):  2, 3, 4, 5, 6, 7, 8, 9
   array[i]:  1, 1, 0, 1, 0, 1, 0, 1
   ```

3. Für Zahl $z_i = 3$ ($i = 1$), werden alle mehrfache von 3 auf 0 gesetzt

   ```
   Index (i): 0, 1, 2, 3, 4, 5, 6, 7
   Zahl (z_i):  2, 3, 4, 5, 6, 7, 8, 9
   array[i]:  1, 1, 0, 1, 0, 1, 0, 0
   ```

4. Die Zahlen $4, 6, 8, 9$ werden übersprungen, da sie schon auf 0 gesetzt sind.
5. Die Zahlen 5 und 7 bewirken keine Veränderungen keine Zahlen da ihr Vielfaches ($10, 15, \ldots$ und $14, 21, \ldots$) kleiner ist als n.
6. Primzahlen sind alle Zahlen die einen Wert von 1 besitzen. Also $2, 3, 5, 7$.

Anders als im Original wird in dieser Variante keine Quadratwurzel gezogen.

1. Aufgabe: Primzahlensieb (3 Punkte)

Schreibe ein Programm, dass alle Primzahlen zwischen 2 und einer eingelesenen Zahl n, einschließlich von n, ausgibt. Zu diesem Zweck soll das Sieb des Eratosthenes (siehe oben) verwendet werden. (Die Zahl 1 ist keine Primzahl.)

Damit das Programm mit beliebig großen Zahlen umgehen kann, soll dynamisch Speicher reserviert werden. Reservierter Speicher muss wieder freigegeben werden. Ein beispielhafter Aufruf inkl. Ausgabe des Programm wird in Listing 1 gezeigt.

Listing 1: Programmbeispiel

```
1 > gcc -std=c99 -Wall ckurs_blatt06_aufgabe01.c input2.c
2                -o ckurs_blatt06_aufgabe01_loesung
3 > ./ckurs_blatt06_aufgabe01
4 Bitte gebe eine Nummer ein: 10
5 2, 3, 5, 7,
```

Wie Du in Listing 1 sehen kannst kompilieren wir die Aufgabe wieder mit einer Bibliothek `input2.c`. Diese Bibliothek stellt die folgenden Funktionen zur Verfügung:

- **`int`** `lese_int()`
  Diese Funktion gibt eine eingelesene Zahl zurück.
- **`void`** `print_prim(`**`int`** `*array,` **`int`** `laenge)`
  Diese Funktion gibt die Primzahlen im Array `array` von der Länge `laenge` aus.

Stelle sicher, dass die Dateien `input2.c` und `input2.h` im selben Ordner liegen.

Um die Hausaufgabe zu vereinfachen bitten wir Dich die vorgegebene Programmstruktur zu verwenden (siehe Listing 2). Den Fall, dass $n = 0$ musst Du nicht gesondert behandeln. Die Abgabe muss folgenden Kriterien entsprechen:

- Die Zahl n wird mithilfe von `lese_int` eingelesen.
- Die Primzahlen zwischen 2 und n, einschließlich von n, werden ausgegeben. Dafür muss die Funktion `print_prim` verwendet werden.
- Die Datei `input2.c` darf nicht verändert werden.
- Der Speicher für das Array mit $n - 1$ Feldern des Typs **`int`** wird mithilfe von `malloc` reserviert.
- Dynamisch reservierter Speicher wird wieder freigegeben.
- Es werden keine zusätzlichen Bibliotheken verwendet.

Listing 2: Mögliche Programmstruktur

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "input2.h"
4
5  int main() {
6      int n = lese_int();
7      int laenge = n-1;
8
9      // Hier implementieren
10
11     // Mit print_prim Primzahlen ausgeben
12     //print_prim(array, laenge);
13
14     return 0;
15 }
```

Checke die Abgabe im SVN ein, wie unter “Abgabemodalitäten” beschrieben.