Stefan Kurtz **Universität Hamburg**
Mathias von Behren **Zentrum für Bioinformatik**

Programmierung in der Bioinformatik
Wintersemester 2015
Übungen zur Vorlesung: Ausgabe am 16.11.2015

Punktevergabe:

- Aufgabe 6.1: 3 Punkte
- Aufgabe 6.2: 2 Punkte
- Aufgabe 6.3: 5 Punkte

Aufgabe 6.1 Betrachten Sie folgenden C-Code:

```
char c,
     *string,
     string2[3] = {0},
     **strings;
int num,
    *nump,
    a,
    *b;

string = "Hallo Welt";
c = string[6];

string2[0] = 'A';
string2[1] = 'B';

strings = malloc(sizeof (*strings) * 3);
strings[0] = &c;
strings[1] = string2;
strings[2] = "third string";

a = 7;
nump = &a;
*nump = 5;
b = &a;

num = (int) (*(strings + 1))[1];
return 0;
```

- Benennen Sie den Typen jeder deklarierten Variablen.
- Welchen Wert haben folgende Ausdrücke: (Wenn der Wert eine Speicheradresse ist, schreiben Sie „Addr“, falls der Wert ein Ascii-Code ist, schreiben Sie z.B. 97: `'a'`)
  - `c`
  - `string2`
  - `string2[0]`
  - `*strings`

  - `strings[2][2]`
  - `num`
- sind folgende Behauptungen wahr?
  - `*strings[0] == *(string + 6)`
  - `b == nump`
  - `*b == *nump`
  - `string == strings[1]`

Aufgabe 6.2 Sie kennen aus der Vorlesung bereits die Funktion `show_sorted`, die zwei sortierte Listen übergeben bekommt und diese als eine einzige aufsteigend sortierte Liste ausgibt.

Schreiben Sie eine Funktion `show_sorted_reverse`, die ebenfalls zwei aufsteigend sortierte Listen und deren Längen übergeben bekommt und diese in absteigender Reihenfolge ausgibt.

Testen Sie ihre Funktion mit den Eingaben

```
[0,3,5,12,16,35]
```

und

```
[1,3,13,15,22,34,42].
```

Aufgabe 6.3 Eine Primzahl ist eine ganze Zahl ≥ 2, die sich nicht ganzzahlig durch eine kleinere Zahl teilen läßt. Implementieren Sie ein C-Programm `erastosthenes.x`, das für eine Zahl n alle Primzahlen berechnet, die kleiner oder gleich n sind. Benutzen Sie zur Berechnung der Primzahlen das „Sieb des Erastosthenes“. Dieses funktioniert wie folgt:

Man schreibt zunächst alle Zahlen i, $2 \leq i \leq n$ auf. Zum Beispiel ergibt sich für $n = 20$ die folgende Zahlenfolge

```
 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

Im nächsten Schritt werden alle echten Vielfachen von 2 markiert:

```
 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
    x     x     x     x     x     x     x     x     x
```

Dann nimmt man die kleinste unmarkierte Zahl ≥ 2 (in diesem Fall 3), und markiert alle echten Vielfachen von 3:

```
 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
    x     x     x  x  x     x     x  x  x     x     x
```

In jedem Schritt werden also jeweils die echten Vielfachen der kleinsten noch unmarkierten Zahl markiert. Das wird solange wiederholt bis die kleinste unmarkierte Zahl größer als $n/2$ ist. Die noch nicht markierten Zahlen sind dann die gesuchten Primzahlen. Beantworten Sie mit Hilfe ihres C-Programms die folgenden Fragen: Wie viele Primzahlen ≤ 1000 gibt es? Was sind die 5 größten Primzahlen ≤ 1000?

Für Ihre Implementierung benutzen Sie bitte ein Array `marked`, das wie folgt in `main` deklariert wird:

```
bool marked[MAXNUMBER+1];
```

Dabei ist `MAXNUMBER` eine Konstante, die durch eine `define`-Anweisung festgelegt wird, und den höchsten Wert von n angibt, den Ihr Programm verarbeiten kann. Der i-te Eintrag des Arrays `marked`, geschrieben als `marked[i]`, enthält genau dann den Wert `false`, wenn die Zahl i bereits markiert ist, und sonst den Wert `true`. Dabei ist `bool`, `true` und `false` in der Standard-Headerdatei `stdbool.h` definiert. Damit Sie dies Verwenden können fügen Sie am Anfang der Sourcedatei die `include`-Anweisung **`#include`** `<stdbool.h>` ein.

Die Lösungen zu diesen Aufgaben werden am 30.11.2016 besprochen.