FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

Algorithmen & Datenstrukturen ILV
Einführung in C

Abgabe: Details zu den Abgabemodalitäten finden Sie auf Moodle. Voraussetzung für die Bewertung ist, dass das Programm kompiliert und ausführbar ist. Die Lösungen werden jeweils am Anfang der nächsten (online) Übungseinheit von zufällig gewählten Studierenden vorgestellt.

Ziele:

- Funktionen
- Pointer / Arrays

Hinweise zum Autograding

Die Übungsaufgaben werden in *main.cpp* gelöst. Alle anderen Dateien bleiben unberührt! Zeile 2 in *main.cpp* - #define TEST 1 - bestimmt ob Tests ausgeführt werden, oder nicht. 0 bedeutet, dass nicht getestet wird. Für die Abgabe muss die Konstante immer den Wert 1 aufweisen.

Entfernen Sie vor der automatischen Bewertung **sämtliche Eingabeaufforderungen (scanf, gets,...)** von der Konsole. Diese führen zu einem Timeout und verhindern das Autograding auf Github Classroom.

Übungsaufgabe 1

Sieb des Eratosthenes
(6 Classroom Punkte)

Eine Primzahl ist eine natürliche Zahl, die nur durch sich selbst und 1 dividierbar ist. Die Brute Force Variante Primzahlen zu berechnen besteht darin, jede einzelne Zahl x bis zu einer Zahl n durchzutesten ob x prim ist (Laufzeit O(n^2) bzw. O($n\sqrt{n}$), Notation wird noch besprochen, nur der Vollständigkeit angeführt). Eine etwas effizientere Möglichkeit besteht darin alle Primzahlen mit Hilfe des "Sieb des Eratosthenes" zu berechnen (Laufzeit O($n\ log\ n$)) (Möhring, 2008).

"Erstellt man eine Liste aller natürlichen Zahlen 1 < n ≤ N und streicht alle echten Vielfachen der Primzahlen p ≤ √N, so verbleiben am Ende genau alle Primzahlen p ≤ N. Obwohl die Idee als Primzahltest denkbar ungeeignet ist, ist das Sieb des Eratosthenes immer noch der effizienteste Algorithmus zur Erstellung einer vollständigen Primzahlliste. Für Zwecke der Erzeugung weniger Primzahlen - wie etwa in der Kryptographie - ist das Sieb des Eratosthenes alles andere als effektiv: Es liefert viele überflüssige Informationen, nämlich die Primfaktorzerlegungen aller natürlichen Zahlen n ≤ N (und in der Tat ist das Sieb des Eratosthenes mehr ein Faktorisierungsalgorithmus denn ein Primzahltest)." (Steuding, 2004). Zur Veranschaulichung dient das unten verlinkte YouTube Video.

Implementierung:

Gegeben sei der Prototyp `void eratosthenes(int n, int* sieve)`. Dabei bezeichnet *n* die Größe des übergebenen Arrays *sieve*. Der Index des Arrays wird in Folge als natürliche Zahl interpretiert.
Das bedeutet, dass das Sieb alle natürlichen Zahlen von 0 bis einschließlich n - 1 auf prim untersucht. Soll bis zur Zahl 100 auf prim getestet werden, ist ein Array der Größe 101 zu übergeben (Index!).

Zu Anfang wird davon ausgegangen, dass alle Zahlen Primzahlen sind (Werte werden auf 1 gesetzt). Für 0 und 1 gilt, dass sie keine Primzahlen sind. Der Algorithmus beginnt folglich wie im Video veranschaulicht bei der Zahl 2 auszusieben. Dabei werden die Vielfachen von 2 gebildet und als Index für das Array *sieve* herangezogen, um die Werte an den entsprechenden Stellen auf 0 zu setzen. Danach wird mit der nächsten Primzahl ausgesiebt.

Auf der englischen Wikipedia Seite https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes wird der Algorithmus samt Optimierungen sehr gut im Abschnitt *Overview* beschrieben.

Beispiele:
(Text in rot = Benutzereingabe)
Eine mögliche Variante der Konsoleneingabe und -ausgabe für eigene Testzwecke (nach 10 Primzahlen folgt ein Zeilenumbruch) in der *main* Methode:

```
n = 20
2 3 5 7 11 13 17 19
```

```
n = 100
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
```

Video:

https://www.youtube.com/watch?v=V08g_lkKj6Q

Literaturquellen:

Steuding, Jörn & Weng, Annegret (2004): Primzahltests - von Eratosthenes bis heute. Mathematische Semesterberichte. Online verfügbar.

Möhring, Rolf und Oellrich, Martin (2008): Das Sieb des Eratosthenes: Wie schnell kann man eine Primzahlentabelle berechnen? Taschenbuch der Algorithmen. Online verfügbar.

Übungsaufgabe 2

Needle in a Haystack
(6 Classroom Punkte)

Implementieren Sie eine Funktion

```
void stringString(char *haystack, char *needle, char **ptr);
```

, welche das erste Vorkommnis des Substrings *needle* in *haystack* findet und die Adresse auf den Beginn des Substrings in *haystack* in die Variable *ptr* schreibt. Die Terminierungssymbole '\0' werden nicht

verglichen. Wird der Substring nicht gefunden, so wird NULL in die Variable *ptr* geschrieben. Funktionen aus der C Standardbibliothek **dürfen nicht** verwendet werden!

Hinweis: Bei der Umsetzung der Aufgabe können Sie eine Brute-Force Methode mit zwei geschachtelten Schleifen anwenden.

Rechercheaufgabe/Diskussion: Welche Aufgaben können von Double Pointern übernommen werden? Wo werden diese eingesetzt?

Übungsaufgabe 3

Min-Max
(3 Classroom Punkte)

In einer quadratischen Matrix aus double Werten soll das Minimum und das Maximum mittels der Funktion minMax gefunden werden. Die Signatur dieser Funktion sieht wie folgt aus:

```
void minMax(double **matrix, size_t size, double **min, double **max);
```

Die Anzahl der Zeilen und Spalten wird mit *size* übergeben. Minimum und Maximum sollen über die Parameter *min* und *max* "zurückgegeben" werden. Allerdings sind hier nicht die Werte zurückzugeben, sondern die Adressen, an denen sich die Werte im Speicher befinden. Testen Sie *minMax* in der *main* Funktion und erzeugen Sie geeignete Testwerte.

Diskussion: Was ist bei der Übergabe von zweidimensionalen Arrays an Funktionen zu beachten? Welche Art der Übergabe liegt im Falle der *minMax* Funktion vor?

Hinweis: überprüfen Sie die Testfunktionen für diese Aufgabe in tests.cpp wie double Pointer initialisiert werden können.

Literaturquellen:

Reese, R. (2013): Understanding and Using C Pointers. O'Reilly, Sebastopol.