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

Inversion Count
(6 Classroom Punkte)

Eine Inversion liegt vor, wenn a[i] > a[j], wobei i < j, also wenn nach einem Wert im Array a noch ein kleinerer auftritt. Das sortierte Array hat 0 Inversionen (jedem Wert folgen nur noch größere). Das invertierte Array hat die maximale Anzahl an Inversionen (jedem Wert folgen nur noch kleinere). Die Anzahl an Inversionen ist ein Maß dafür wie weit ein Array vom sortierten Zustand entfernt ist.

Gegeben sei der Prototyp `int inversionCount(int size, int *numbers, int *inversions)`. Der Parameter *size* bestimmt die Größe der beiden übergebenen Arrays *numbers* und *inversions*. Hinweis: Für beide Arrays wurde bereits Speicher in der aufrufenden Funktion allokiert. Der Rückgabewert vom Typ *int* soll die Gesamtanzahl der vorliegenden Inversionen im Array *numbers* zurückliefern. Implementieren Sie den Algorithmus basierend auf der obigen Definition. Die Beispiele unten verdeutlichen die Vorgangsweise.

Beispiele:
(Text in rot = Benutzereingabe)
Dienen der besseren Veranschaulichung. Sie können diese freiwillig in der *main* Methode nachbauen. Hier:

- liest zunächst die Anzahl N der einzulesenden Werte
- liest N Werte in ein Array
- gibt die Anzahl an Inversionen aus ("%d inversions.\n")
- gibt das Array aus ("%2d ")
- gibt die Anzahl an Inversionen für jedes Element im Array aus ("%2d ")

```
9
5 6 3 5 7 8 9 1 2
17 inversions.
5 6 3 5 7 8 9 1 2
3 4 2 2 2 2 2 0 0
```

```
5
1 2 3 4 5
0 inversions.
1 2 3 4 5
0 0 0 0 0
```

Übungsaufgabe 3

Memory Swap
(3 Classroom Punkte)

Schreiben Sie eine Funktion *memswap*, die zwei beliebige Speicherbereiche gleicher Größe austauscht. Die Funktion soll zwei Zeiger vom Typ „char *" auf die beiden auszutauschenden Speicherbereiche und einen „int" Wert entgegennehmen, der die Größe der Speicherbereiche in Bytes angibt. („char *" ist geeignet, da der Typ „char" genau 1 byte groß ist). Die Funktion soll die folgende Signatur besitzen:

```
void memswap(char *mem1, char *mem2, int size);
```

Testen Sie die Funktion beispielsweise in *main* mit zwei *int* oder *double* Arrays. Initialisieren Sie diese vorab mit konstanten Werten.

FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

Beispiele:
(hier zwei int arrays)

```
vor Memswap:
1: 12 15 22 44 100 3000
2: 1 2 3 4 5 6
nach Memswap:
1: 1 2 3 4 5 6
2: 12 15 22 44 100 3000
```