# Primzahlen

**Allgemeine Anmerkungen zur Aufgabenstellung:**

- Beantworten Sie die Kontrollfragen in `TUWEL` bevor Sie mit der Übung beginnen.
- Legen Sie für `Teil A` der Angabe ein neues `Code::Blocks`-Projekt an und erweitern Sie dieses schrittweise für die darauffolgenden Punkte.
- Stellen Sie während der Übung laufend sicher, dass Ihr Programm kompilierbar ist und richtig funktioniert.
- Die Abgabe erfolgt durch Hochladen Ihres vollständigen `Code::Blocks`-Projektordners als zip-Datei (`Matrikel-Nr_UE3.zip`) in `TUWEL`. Anschließend melden Sie sich für das Abgabegespräch in `TUWEL` an. Dieses erfolgt über die Telekonferenz-App `Zoom` (www.zoom.us). Stellen Sie sicher, dass diese App rechtzeitig vor Beginn des Abgabegespräches auf Ihrem PC installiert ist und dass Sie Ihr aktuelles Übungsprogramm in `Code::Blocks` für die Abgabe geöffnet haben. Testen Sie bitte auch die korrekte Funktionalität Ihrer Videokamera, da Sie zu Beginn Ihres Abgabegespräches Ihren Studentenausweis herzeigen müssen und auch während des Gesprächs eine aktive Videoverbindung erforderlich ist.
- Beachten Sie, dass Upload und Terminauswahl für das Abgabegespräch erst möglich sind, wenn Sie 80% der Kontrollfragen richtig beantwortet haben. Bis **04.05.2021 23:59** müssen diese Fragen positiv beantwortet, Ihre Ausarbeitung hochgeladen und ein Termin für das Abgabegespräch ausgewählt werden.
- Inhaltliche Fragen stellen Sie bitte in der Fragestunde oder im zugehörigen Forum in `TUWEL`.
- Für eine positive Beurteilung Ihrer Abgabe **muss** diese kompilierbar sein, Ihr Programm fehlerfrei terminieren und die in der Angabe angeführten Funktionen mit den vorgegebenen Funktionsköpfen enthalten. Des Weiteren müssen die Funktionen hinreichend getestet sein. Sie müssen in der Lage sein, Ihr Programm beim Abgabegespräch zu erklären!

**Folgende Hilfsmittel stehen Ihnen in `TUWEL` zur Verfügung:**

- Alle bisherigen Referenzbeispiele sowie Vorlesungsfolien
- Interaktive Jupyter-Notebooks auf `prog1.iue.tuwien.ac.at`
- Das Buch zur LVA "Programmieren in C" (Robert Klima, Siegfried Selberherr)
- Kurzanleitung für die Entwicklungsumgebung `Code::Blocks`

**Hinweis:** Es wird erwartet, dass alle Abgaben zu den Übungen **eigenständig** erarbeitet werden und wir weisen Sie darauf hin, dass alle Abgaben in einem standardisierten Verfahren auf Plagiate überprüft werden. Bei Plagiatsvorfällen entfällt das Abgabegespräch und es werden keine Punkte für die entsprechende Abgabe vergeben.

**Einleitung**

Primzahlen spielen in vielen Verschlüsselungsalgorithmen eine bedeutende Rolle. In dieser Übung werden Sie zunächst einen einfachen Primzahl-Test implementieren um anschließend eine Primfaktor-Zerlegung für beliebige Zahlen durchführen zu können. Dabei werden Sie den Umgang mit Schleifen und Funktionen in der Sprache C erlernen und üben.

Zusätzlich zum Stoff aller vorangegangenen Übungen benötigen Sie für diese Übung folgende neue Themengebiete aus den Vorlesungen 4-6 (Kapitel 10-13):

- Schleifen/Iterationen: `for, do while, while` (Kapitel 10)
- Befehle: `break, continue` (Kapitel 10.5, 10.6)
- Funktionen (Kapitel 11)
- Speicherklassen: `auto, static` (Kapitel 12.1, 12.2)
- Felder (Kapitel 13.1, 13.3)

## Teil A

✘ Implementieren Sie eine Funktion

```
long readNumber();
```

welche eine ganze Zahl von der Tastatur einliest und zurückgibt. Fangen Sie dabei eventuell auftretende fehlerhafte Eingaben direkt in der Funktion ab!

✘ Schreiben Sie eine Funktion

```
long isPrime(long num);
```

die feststellt, ob es sich beim übergebenen Argument `num` um eine Primzahl handelt. Die Funktion soll folgende Rückgabewerte liefern:

- $-1$, wenn `num` $\leq 0$
- $0$, wenn `num` *keine* Primzahl ist
- $+1$, wenn `num` *eine* Primzahl ist

**Hinweis:** Eine Primzahl ist eine natürliche Zahl größer als 1, die nur durch 1 und sich selbst teilbar ist. Sollten Sie Probleme bei der Implementierung haben, führen Sie eine Primzahlprüfung am Papier durch; lassen Sie dabei keine Zwischenschritte aus! Versuchen Sie anschließend, aus den durchgeführten Schritten Programmanweisungen zu formulieren.

✘ Schreiben Sie ein Hauptprogramm, das wiederholt eine Zahl eingeben lässt und bestimmt ob diese Zahl prim ist. Vergessen Sie dabei nicht eine Abbruchbedingung zu definieren. Ein möglicher Programmablauf könnte wie folgt aussehen:

```
Geben Sie eine Zahl ein: 12
12 ist nicht prim
-------------------------
Nochmal (y|n)? y

Geben Sie eine Zahl ein: 5   \
5 ist prim
-------------------------
Nochmal (y|n)? n

------PROGRAMM-ENDE-------
```

## Teil B

✘ Schreiben Sie eine Funktion

```
void primeFactors(long num);
```

die die Primfaktoren ihres Arguments bestimmt und auf dem Bildschirm ausgibt. Die Ausgabe soll dabei *genau* folgende Form haben:

```
12 = 2 * 2 * 3
```

✘ Erweitern Sie Ihr Hauptprogramm so, dass für die eingegebene Zahl zusätzlich die Primfaktor-Zerlegung ausgegeben wird, sofern diese keine Primzahl ist:

```
Geben Sie eine Zahl ein: 12
12 ist nicht prim
Primfaktor-Zerlegung: 12 = 2 * 2 * 3
-------------------------
```

```
Nochmal (y|n)? y

Geben Sie eine Zahl ein: 5
5 ist prim
-------------------------
Nochmal (y|n)? n

------PROGRAMM-ENDE-------
```

## Teil C

✘ Implementieren Sie zusätzlich zu `isPrime` eine weitere Funktion

```
long isPrimeFast(long num);
```

welche ebenfalls einen Primzahl-Test durchführt. Allerdings soll beim *ersten Aufruf* ein Feld vorbereitet werden, welches für jede Zahl kleiner 100 speichert, ob sie prim ist. Bei jedem weiteren Aufruf wird der Rückgabewert nur noch mit Hilfe dieses Feldes bestimmt, falls die übergebene Zahl kleiner als 100 ist. Für größere Zahlen soll die Primzahl-Überprüfung wie gewohnt durchgeführt werden. Vermeiden Sie für diese Erweiterung die Verwendung globaler Variablen und benutzen Sie `isPrime` an geeigneten Stellen wieder.

✘ Es wird vermutet, dass alle geraden Zahlen größer 2 als Summe zweier Primzahlen darstellbar sind (Goldbach'sche Vermutung). Schreiben Sie eine Funktion

```
void expandNumber(long num);
```

die eine solche Zerlegung für eine übergebene Zahl durchführt und ausgibt:

```
12 = 5 + 7
```

**Hinweis:** Gehen Sie dabei alle Möglichkeiten durch, die Zahl als Summe darzustellen und ermitteln Sie, ob es sich bei den Summanden um Primzahlen handelt.

✘ Erweitern Sie Ihr Hauptprogramm so, dass für gerade Zahlen größer 2 zusätzlich eine mögliche Goldbach-Zerlegung ausgegeben wird:

```
Geben Sie eine Zahl ein: 48
48 ist nicht prim
Primfaktor-Zerlegung: 12 = 2 * 2 * 2 * 2 * 3
Goldbach-Zerlegung:   48 = 19 + 29
-------------------------
Nochmal (y|n)? y

Geben Sie eine Zahl ein: 5
5 ist prim
-------------------------
Nochmal (y|n)? n

------PROGRAMM-ENDE-------
```