Arbeitsblatt: INF1

Name: **Kurznamen:**

Funktionen

Aufgabe 1: Rechtwinkliges Dreieck

Es soll eine Funktion ***bool rechtwinklig (double a, double b, double c)*** geschrieben werden, die überprüft, ob die als Parameter übergebenen Seitenlängen ein rechtwinkliges Dreieck ergeben. Nach der Eingabe soll die Funktion mit den (in der main Funktion) eingelesenen Werten aufgerufen und das Resultat (wieder in der main Funktion) ausgegeben werden. Die Trennung zwischen Verarbeitung von Werten und Input/Output ist ein wichtiger Grundsatz bei der Programm Strukturierung.

Überlegen Sie sich, wieso wir die Eingabe (der drei Werte) nicht ebenfalls als eine Funktion schreiben können.

Abgabe

Praktikum: INF5.1

Filename: triangle.c

Aufgabe 2: Fibonacci-Zahlen

Die Fibonacci-Zahlen bilden eine Zahlenfolge, die sich rekursiv folgendermassen definiert:

$$F_n = \begin{cases} 0 & \text{für} \quad n = 0 \\ 1 & \text{für} \quad n = 1 \\ F_{n-1} + F_{n-2} & \text{für} \quad n > 1. \end{cases}$$

Der dritte Teil der Definition besagt, dass sich Fibonacci-Zahlen ab der dritten aus der Summe der beiden aufeinander folgenden Vorgänger ergeben.

n	0	1	2	3	4	5	6	7	8	9	...
F_n	0	1	1	2	3	5	8	13	21	34	...

Implementieren Sie eine *rekursive* Funktion ***int fibonacci(int a)****, i.e.* eine Funktion, die sich selber aufruft.

Abgabe

Praktikum: INF5.2

Filename: fibonacci.c

Aufgabe 3: Testgetriebene Entwicklung

Unter der Bezeichnung *Testgetriebene Entwicklung* (engl. *test-driven development*, TDD, auch: *test first development*) versteht man eine Vorgehensweise bei der Software-Entwicklung, bei der Tests konsequent vor den zu testenden Komponenten erstellt werden. Die scheint zwar zunächst mehr Aufwand zu sein, hat aber eine Reihe von Vorteilen:

- Man ist gezwungen, sich schon vor der Entwicklung der eigentlichen Komponenten mit den erwarteten Resultaten und möglichen Problemfällen zu beschäftigen.
- Jederzeit können alle Testfälle automatisiert geprüft werden, um schnell einen Überblick über den Stand der Entwicklung zu erhalten.
- Die Testfälle können als Teil der Dokumentation des Systems aufgefasst werden, da sie das gewünschte Verhalten beschreiben.
- Ausser bei einfachen Problemen erreicht man selten mit dem ersten Anlauf zu einer Problemstellung eine perfekte Lösung. Daher wird es oft nötig, einmal entwickelte Komponenten eines grösseren Systems immer wieder umzubauen – nicht ganz im Sinne des verbreiteten Spruchs: *„Never touch a running system."* Wenn jedoch umfassende Tests für eine Komponente bereitstehen, wird dieses Umbauen (*Refactoring*) wesentlich weniger fehleranfällig. Man kann ja jederzeit die Tests laufen lassen und stellt so zum Beispiel schnell fest, wenn beim Umbau irgendwelche Ausnahmefälle vergessen wurden.

Der Nutzen der *Testgetriebenen Entwicklung* steht und fällt natürlich mit der Qualität der Tests. Normalerweise werden für das Erstellen und Durchführen der Tests geeignete Bibliotheken oder Frameworks verwendet. In vereinfachter Form soll eine solche Vorgehensweise nun mit der Rechtwinklig-Funktion demonstriert werden.

Für den Test der Rechtwinklig Funktion schreiben wir eine Funktion *rechtwinkligTest*. Erfolg oder Misserfolg der Tests wird direkt von der Funktion auf der Konsole gemeldet. Fügen Sie Tests für die Fibonacci-Zahlen hinzu.

```
#include "assertions.c"
#include <stdio.h>
#include <math.h>
#include <stdbool.h>

/* vorwärtsdeklarationen */
bool rechtwinklig(double a, double b, double c);
int fibonacci(int a);

void rechtwinkligTest () {
 /* Tests */
  assertEquals("3,4,5",true,rechtwinklig(3,4,5));
  assertEquals("5,3,4",true,rechtwinklig(5,3,4));
  assertEquals("8.5,4,7.5",true,rechtwinklig(8.5,4,7.5));
  assertEquals("8.8,10.5,13.6",false, rechtwinklig( 8.8, 10.5, 13.6));
}

void fibonacciTest () {
 /* Tests */
}

int main (void) {
      rechtwinkligTest ();
      fibonacciTest ();
      return 0;
}
```