Arbeitsblatt 6. Funktionen

Aufgabe 6.1. Ausgabe der Zahlen mit der Ziffer "7"

Grundlage für diese Aufgabe ist die in der Vorlesung entwickelte Funktion „EnthaeltZiffer“ (siehe Git).

a) Modifizieren Sie die Funktion so, dass Sie feststellen können, ob die Zahl die Ziffer **genau einmal** enthält. Ergänzen Sie zur Steuerung dieser Funktionalität einen weiteren Parameter vom Typ bool, so dass Ihre Funktion beides kann. Machen Sie diesen Parameter zu einem Default-Parameter mit Default-Wert false.

b) Schreiben Sie ein Hauptprogramm, dass zwei ganze Zahlen a und b erfragt und alle Zahlen von a bis einschließlich b ausgibt, welche die Ziffer "7" genau einmal enthalten.

Aufgabe 6.2. Exchange-Funktion (Call-by-Reference)

a) Schreiben Sie eine Funktion "**Exchange**", die zwei **Double**-Werte als Parameter bekommt und dafür sorgt, dass die Werte vertauscht werden. Verwenden Sie das hier definierte Hauptprogramm als Test (Achtung, es fehlt noch etwas!):

```
double zahl1 = 17;
double zahl2 = 23;

Console.WriteLine($"zahl1 = {zahl1}, zahl2 = {zahl2}");
Exchange(zahl1, zahl2);
Console.WriteLine("Exchange!");
Console.WriteLine($"zahl1 = {zahl1}, zahl2 = {zahl2}");
```

Das Programm soll diese Ausgabe liefern:

```
zahl1 = 17, zahl2 = 23
Exchange!
zahl1 = 23, zahl2 = 17
```

b) Integers werden normalerweise automatisch in Double konvertiert, wenn nötig.
Warum können Integer-Variablen nicht in Double-Ref-Parameter konvertiert werden?
Falls Sie es nicht wissen, fragen Sie mal ChatGPT.

c) Optional: Ergänzen Sie die Funktion so, dass die Zahlen zahl1 und zahl2 erfragt werden. Stellen Sie über Double.TryParse() und eine entsprechende Schleife sicher, dass nur Zahlen eingegeben werden dürfen.

Aufgabe 6.3. Rekursive Potenzfunktion

In Aufgabe 5.1 wurde eine Potenzfunktion entwickelt. Nutzen Sie Ihre Lösung von letzter Woche oder die Musterlösung als Basis für diese Aufgabe.

Implementieren Sie eine rekursive Variante "PotenzRek(x, n)" der Potenz und testen Sie sie.

Es gilt: $$x^n = \begin{cases} x \cdot x^{n-1} & falls\ n > 0 \\ 1 & falls\ n = 0 \end{cases}$$

Testen Sie die Funktion, z.B. mit $2^3 = 8$, im Debugger, damit Sie sehen, was passiert.

Aufgabe 6.4. Wurzelberechnung nach dem Heron-Verfahren (ohne Feld)

Das Heron-Verfahren ist ein iteratives Verfahren zur näherungsweisen Bestimmung der Quadratwurzel einer Zahl. Die Iterationsvorschrift lautet:

$$w_{i+1} = \frac{1}{2} \cdot \left(w_i + \frac{x}{w_i} \right)$$

Es bezeichnet x die Zahl, deren Quadratwurzel bestimmt werden soll und w_i den Wert den approximierten Wert der Wurzel von x in Iteration i. Der Startwert w_0 kann eine beliebige Zahl > 0 sein, z.B. x/2.

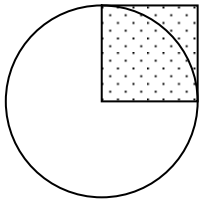
Erstellen Sie eine Funktion `double WurzelHeron(double x)` zur Berechnung der Quadratwurzel. Das obige Iterationsverfahren soll stoppen, wenn der quadrierte Wert der ermittelten Wurzel w nur noch höchstens 1.0^{-12} vom Ausgangswert x abweicht, d.h. wenn

```
Math.Abs(x - w*w) > 1.0e-12
```

Natürlich darf Math.Sqrt() für diese Funktion nicht verwendet werden. Sie dürfen Ihr Ergebnis aber gern mit dem von Math.Sqrt() vergleichen.

Aufgabe 6.5. Bestimmung von Pi mit der Monte-Carlo-Methode

Bestimmen Sie die Zahl Pi mittels der Monte Carlo-Methode. Hierbei werden jeweils zwei Zahlen x und y im Bereich von [0..1] gewürfelt. Dieses Zahlenpaar entspricht einer Koordinate im Einheitsquadrat. Für jedes Zahlenpaar ist nun zu testen, ob sich die Koordinate innerhalb oder außerhalb des 1. Quadranten des Einheitskreises befindet. Dieser Test kann leicht mittels einer Abstandsberechnung (Pythagoras!) vorgenommen werden.



Das Verhältnis der Punkte im Einheitskreis zur Gesamtanzahl der gewürfelten Punkte entspricht der Zahl Pi/4 (Warum?). Lassen Sie hinreichend viele Zahlen würfeln, um Pi mit zufriedenstellender Genauigkeit zu erhalten. Verpacken Sie Ihren Algorithmus in eine Funktion, welcher die Anzahl der Zufallszahlen als Parameter erhält. Default soll 100 sein.

Hinweis: Zum Würfeln einer Gleitkommazahl können Sie die NextDouble-Funktion der Klasse Random verwenden. Diese liefert Gleitkommazahlen im Bereich von [0.0 … 1.0]:

```
Random zufall = new Random(); // Anlegen eines Würfels; einmalig!
double x = zufall.NextDouble(); // Würfeln der nächsten Zufallszahl
```

Aufgabe 6.6. Funktion zur Ausgabe eines Dreiecks

a) Schreiben Sie eine Funktion ZeichneDreieck, die als Parameter eine ganze Zahl n sowie ein Zeichen c akzeptiert und ein n-zeiliges Dreieck mit Sternchen auf den Bildschirm zeichnet!

```
ZeichneDreieck(6, '*');
*
**
***
****
*****
******
```

b) Machen Sie c zu einem optionalen Parameter, default '*'

c) Ergänzen Sie einen weiteren Parameter, centered (default false), der angibt, ob das Dreieck zentriert ausgegeben werden soll

```
ZeichneDreieck(5, '#', true);
    #
   ###
  #####
 #######
#########
```