Übungen zur Arithmetik (1)

1. Beispiele zur Veranschaulichung der automatischen Typkonvertierung:
   Vergleichen Sie die Ergebnisse verschiedener arithmetischer Ausdrücke bei unterschiedlicher Schreibweise der Konstanten als ganze oder „reelle“ Zahlen

   ```
   ( 2/3, 2./3, 2d0/3, 1/2*4.0, 1/2.0*4,  usw. ),
   ```

   sowie bei Zuweisung des Ergebnisses an Variablen unterschiedlichen Typs (`INTEGER` oder `REAL`).
   Werden die Ergebnisse bei Zuweisung an eine Integer-Variable korrekt gerundet oder wird nur der gebrochene Anteil abgeschnitten? – Überprüfen Sie dies anhand geeigneter Ausdrücke mit positiven und negativen Ergebnissen.

2. **Integer-Arithmetik:** Berechnung von $n! = n \cdot (n-1) \cdot \ldots \cdot 1$.
   Schreiben Sie ein Funktionsunterprogramm (`integer function nfak(n)`), welches dieses Produkt innerhalb einer Schleife berechnet.
   Hinweis zu Funktionsunterprogrammen in Fortran:

   ```
   integer function nfak(n)
   ...
   nfak = <ausdruck>       ! Funktionswert zurueckgeben
   return
   end
   ```

   Der Funktionsname (`nfak`) darf (muss aber nicht) innerhalb der Funktion beliebig als lokale Variable benutzt werden, ihm muss aber am Ende der Funktionswert zugewiesen werden.

   Testen Sie zunächst für kleine `n`, ob das Programm richtige Ergebnisse liefert.

   Zur Darstellung positiver ganzer Zahlen werden 31 Bit benutzt: $\sum_{k=0}^{30} b_k 2^k,\ b_k \in \{0,1\}$.
   Welche Reaktion ist bei „zu großem“ n zu erwarten? Testen Sie, ob die erwartete Reaktion auch wirklich eintritt?

   Vergleichen Sie die Ergebnisse der `Integer`-Arithmetik mit denen von `Real` und `DoublePrecision`, indem Sie alle Datentypen (incl. Funktionen) entsprechend ändern, bzw. alle 3 Funktionen gelichzeitig verwenden:

   ```
   integer function nfak(n)
   real function rfak(rn)
   doubleprecision function dfak(dn)
   ```

3. Stellen Sie fest, ob der verwendete Fortran-Compiler rekursive Funktionsaufrufe zulässt (direkt oder indirekt), so dass man $n!$ nach der Rekursionsvorschrift berechnen kann:

   $$n! = \begin{cases} n \cdot (n-1)!, & n > 1 \\ 1, & \text{sonst} \end{cases}$$

   Direkte Rekursion z.B. durch den Aufruf : `nfak = n*nfak(n-1)` in der Funktion.
   Indirekte Rekursion z.B. durch 2 Funktionen

```
integer function nfak(n)
integer n,dummy
...
nfak = n*dummy(n-1)
```

```
integer function dummy(n)
integer n,nfak

dummy=nfak(n)
```

4. Binomialkoeffizienten

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$$

kann man auf verschiedene Arten berechnen, z.B.

- nach obiger Definition mit Hilfe eines Unterprogramms, das $n!$ berechnet, oder aber
- als fortlaufendes Produkt von Quotienten

$$B_j = B_{j-1} * \frac{n+1-j}{j}, \qquad \text{d.h. } \binom{n}{k} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{1 \cdot 2 \cdot \ldots \cdot k}$$

  Darf man dies mit ganzzahliger Arithmetik ausführen?

- Rekursiv durch Addition (vgl. Pascalsches Dreieck)

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

Schreiben Sie je ein Unterprogramm für diese drei Berechnungen und ein Hauptprogramm, in dem die Zahlen n und k eingelesen und die drei berechneten Werte ausgegeben werden.