Cheat Sheet - Modularisierung, Arrays

Aufgabe:

Teil 1

Implementiert ein Modul "LinearAlgebra" mit den Funktionen

`makeIdentityMatrix(int size) => real[,]`
Erzeugt eine Einheitsmatrix der Größe `size`

Beispiel: `size = 2` ergibt

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

`vecToHomogCoords(real[] vector) => real[]`
Erzeugt ein Array der Länge n+1, wobei die ersten n Elemente kopiert werden und das Element n+1 gleich 1 ist

Beispiel für Vektor der Länge 3:

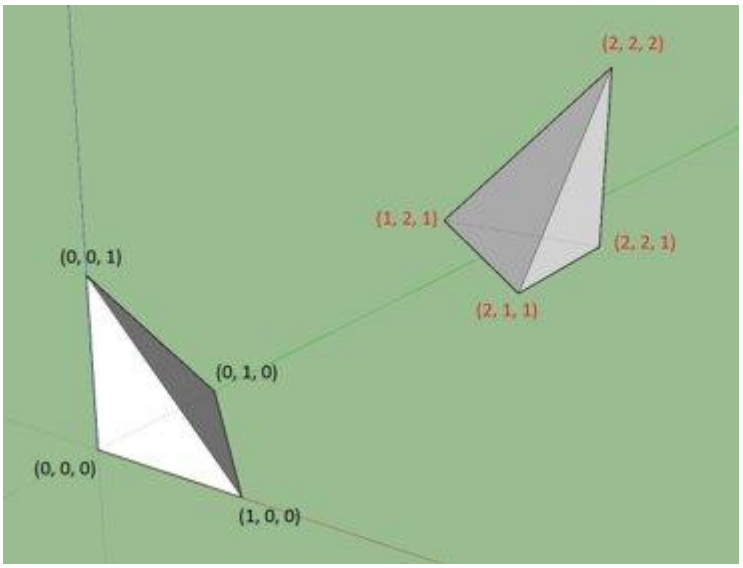
$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

`homogCoordsToVec(real[] coords) => real[]`
Erzeugt ein Array der Länge n-1, wobei die Einträge kopiert und durch das n-te Element dividiert werden

Beispiel für Vektor der Länge 4:

$$f\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{pmatrix}$$

Teil 2



Implementiert mit Hilfe des Moduls ein Programm, das ein Objekt mit Punkten `(0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 1, 1)` wie folgt transformiert:

- Verschiebung um (2, 2, 1)
- Rotation (um die z-Achse) mit θ = 180° (θ = π)

Benutzt dazu die Transformationsmatrix

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & x \\ \sin(\theta) & \cos(\theta) & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Tipps:

- Alle Funktionen aus Aufgabe 1 sind hilfreich für Aufgabe 2
- Die Transformation eines Punktes entspricht einer Matrixmultiplikation der Transformationsmatrix mit dem Ortsvektor
- Die Transformationsmatrix ist eine 4x4 Matrix, die Ortsvektoren haben aber nur 3 Koordinaten. Damit Matrix und Vektor multipliziert werden können, ist zunächst eine Umwandlung in homogene Koordinaten erforderlich

Hilfreiche Codebeispiele

Kompilieren (Reihenfolge beachten)

```
gfortran somelib.f90 program.f90
```

Funktion mit Parametern und Rückgabewert

```
function add(a, b) result(c)
  implicit none
  real, intent(in) :: a, b
  real :: c
  c = a + b
end function add
```

Deklaration von Modulen in eigener Datei

```
module SomeLib
  implicit none
  real, parameter :: pi = 3.14159265
contains
  function add(a, b) result(c)
    ! ...
  end function add
end module
```

Verwendung von Modulen

```
program ModuleExample
  use SomeLib
  implicit none
  real sum
  sum = add(pi, 1.0)
end program
```

Arrays deklarieren

```
real, dimension(3) :: vecA

integer :: a = 5
real, dimension(a) :: vecB

subroutine foo(vecC)
  implicit none
  real, dimension(:), intent(in) :: vecC
end subroutine foo
```

Array-Elemente zuweisen

```
real, dimension(2) :: vecA = 0
real, dimension(2) :: vecB = (/ 0, 2 /)
real, dimension(3) :: vecC = 1
vecC(1:2) = vecB
```

Matrizen, Matrixmultiplikation

```
real, dimension(2, 3) :: M
M(1, 1:3) = (/ 11, 12, 13 /) ! a11 a12 a13
M(2, 1:3) = (/ 21, 22, 23 /) ! a21 a22 a23

write(*, *) M
>>> 11.0   21.0   12.0   22.0   13.0   23.0
```

```
real, dimension(3) :: vec = (/ 1, 1, 1 /)
real, dimension(2, 3) :: M
M = reshape((/ 11, 21, 12, 22, 13, 23 /), shape(M))
matmul(M, vec)
```

Schleifen

```
integer :: i
do i = 1, 4
    !...
end do
```