Ein mathematischer Vektor

In der Mathematik und Physik spielen Skalare und Vektoren eine große Rolle. Während man unter einem *Skalar* eine Größe versteht, deren Wert sich eindeutig durch die Angabe einer *Maßzahl* beschreiben lässt, benötigt man zur vollständigen Charakterisierung eines *Vektors* noch die Angabe einer Richtung im Raum. So lassen sich beispielsweise Windgeschwindigkeiten, Beschleunigungen, elektrische oder magnetische Feldstärken durch Vektoren beschreiben.

Implementieren Sie eine Reihe von Funktionen (auf der Basis von C-Arrays mit Elementen - sprich Richtungskomponenten - vom Typ `double`, die folgende Vektoroperationen bereitstellten:

- Initialisierung eines n-dimensionalen *Nullvektors* – Ein Nullvektor ist ein Vektor, bei dem alle Richtungskomponenten gleich Null sind.
- Betrag eines Vektors – Ein Vektor ist im anschaulichen Sinne eine gerichtete Strecke im Raum. Die Länge dieser Strecke nennt man *Betrag* des Vektors.
- Addition und Subtraktion zweier Vektoren – Liefert als Ergebnis einen Vektor, dessen Komponenten die Summe (Differenz) der entsprechenden Komponenten der Summanden ist.
- Multiplikation eines Vektors mit einem Skalar – Jede Richtungskomponente des Vektors ist mit einer skalaren Größe zu multiplizieren. Im Gegensatz zur Addition ist die Multiplikation eines Vektors mit einem Skalar eine sinnvolle Operation.
- Skalares Produkt zweier Vektoren – Liefert als Ergebnis die Summe der Produkte der jeweiligen Komponenten der beiden Vektoren zurück (Wert vom Typ `double`).
- Kreuz-Produkt zweier Vektoren – Liefert als Ergebnis im dreidimensionalen Fall zu zwei Vektoren wieder einen Vektor zurück, der senkrecht auf der von den beiden Ursprungsvektoren aufgespannten Ebene steht (Ergebnis vom Typ `double[]`).
- Vergleich zweier Vektoren auf Gleichheit oder Ungleichheit.
- Einfache Ausgabe eines Vektors auf der Konsole.

Sie dürfen bei Ihrer Implementierung für die Dimension entweder den Wert 3 oder einen allgemeinen, durch eine `#define`-Direktive festgelegten, Wert zu Grunde legen.

Die folgenden Funktionsprototypen stellen eine Orientierungshilfe für die zu entwickelnden Funktionen dar:

```
// function prototypes
void   VectorInit          (double vector[], int len);
double VectorLength        (double vector[], int len);
void   VectorAdd           (double result[], double vec1[], double vec2[], int len);
void   VectorSub           (double result[], double vec1[], double vec2[], int len);
void   VectorScalarMul     (double result[], double vec[], int len, double scalar);
void   VectorCrossProduct  (double result[], double vec1[], double vec2[], int len)
double VectorScalarProduct (double vec1[], double vec2[], int len);
int    IsEqual             (double vec1[], double vec2[], int len);
int    IsUnequal           (double vec1[], double vec2[], int len);
void   VectorPrint         (double vector[], int len);
```