Testaufgabe: Containerklasse für Dreiecke

Ziel der gestellten Aufgabe ist die Entwicklung einer Containerklasse, die eine bestimmte Anzahl von Dreiecken aufnehmen kann. Dabei wird durch das Verfahren der tiefen Kopie dafür Sorge getragen, dass die im Container enthaltenen Dreiecke unabhängig von den an der Klassen-Schnittstelle übergebenen sind. Insbesondere bedeutet dies, dass die innerhalb des Containers liegenden Dreiecke unabhängig von der Lebenszeit der Dreiecke sind, die an der Schnittstelle übergeben wurden!

Die Dreiecke sind im 2-dimensionalen Raum repräsentiert. Zur Darstellung der Eckpunkte der Dreiecke existiert die Struktur ***coords***:

```
// struct: coords
struct coords {
   coords() : first(), second() {};
   coords(const float _Val1, const float _Val2)
      : first(_Val1), second(_Val2) {};

   float first;
   float second;
};
```

Die Dreiecke selbst werden durch eine Klasse ***triangle*** repräsentiert:

```
// class: triangle
static const int anzTriangleCoords = 3;

class triangle {
   friend std::ostream& operator<<(std::ostream &os, const triangle &_Val);
   friend class triangleContainer;

   coords *_pCoords[anzTriangleCoords];

protected:
   const coords& getCoords(int _Idx) const;

public:
   triangle();
   triangle(const coords &_Val1, const coords &_Val2, const coords &_Val3);
   triangle(const triangle& _Val);
   ~triangle();

   triangle& operator=(const triangle &_Other);
   bool setCoords(int _Idx, const coords &_Val);
};
```

Der Container selbst hat seine Repräsentation in der Klasse ***triangleContainer***, deren Deklaration wie folgt aussieht:

```
// class: triangleContainer
static const int maxTriangles = 100;

class triangleContainer
{
   friend std::ostream& operator<<(std::ostream &os, const triangleContainer &_Val);

   int _length{ 0 };
   triangle *_pTriangleField[maxTriangles];

protected:
   const triangle& getTriangle(int _Idx) const;

public:
   triangleContainer(int _Val);
   triangleContainer(const triangleContainer &_Val);
   ~triangleContainer();

   triangleContainer& operator=(const triangleContainer &_other);
   int length() const;
   bool addTriangle(const triangle &_Val);
   bool changeTriangle(int _Idx, const triangle &_val);
};
```

Als Ausgangspunkt wird ein VS 2019 Projekt zur Verfügung gestellt, das unter anderem die oben angeführten Klassendeklarationen enthält. Zudem werden zur leichteren Ausgabe auf der Konsole überladene OutStream Operator Funktionen zur Verfügung gestellt. Damit deren Signatur im Header File bekannt ist, sind sie dort durch Prototypes vordeklariert.

Ebenso existiert im Projekt eine ***main*** Funktion, die zum Test der zu implementierenden Klassen dient. Achten Sie bitte darauf, dass der Code der vorgegebenen ***main*** Funktion nicht alle zu implementierenden Aspekte der Klassen testet. Gegebenenfalls muss die ***main*** Funktion entsprechend erweitert werden.

Anmerkungen:

1. Für die zur Verfügung gestellten Quelldateien sind die Include Einträge selbst zu erstellen!
2. Die verwendeten ***friend*** Deklarationen ermöglichen den Zugriff auf die sonst nicht sichtbaren Bereiche der „befreundeten“ Klassen und Funktionen.
3. Die Methode ***changeTriangle*** des Containers dient zum Austausch eines Dreiecks im Container und ist so implementiert, dass dazu kein neues Objekt erzeugt wird. Die Methode ist so beizubehalten!