Übung Nr. 3:

Gegenstand dieser Übung ist der Umgang mit grafischen Objekten, die aus formatierten (d.h.: Text- bzw. ASCII-) Dateien geladen und in Strukturen gespeichert werden. Das entsprechende MS-VC-Projekt (**`LoadSetCGF`**) unter http://homepages.thm.de/christ/ enthält die Dateien:

`setCGF.c`	In dieser Datei befindet sich die Funktion **`int setTriangleCGF (CGFobject *obj)`** Mit ihr wird ein Dreieck als Objektstruktur gesetzt. Diese Funktion muß noch vervollständigt werden (ca. 15 Zeilen, die Stellen sind mit `MORE_SET` gekennzeichnet).
`LoadCGF.h`	Header-Datei mit der für diese Vorlesung entworfenen (einfachen, aber praxisnahen) Objektstruktur und den Prototypen der hier vorgestellten Funktionen.
`LoadCGF.c`	C-Datei mit zwei Funktionen, die typischerweise für die Behandlung solcher Objekte gebraucht werden: **`int printCGF (CGFobject *obj)`** gibt die Werte der Struktur-Elemente eines geladenen Objektes aus. Diese Funktion (ca. 20 Zeilen) muß aus dem gelieferten Code zusammengestellt (kopiert) werden. **`int loadCGF (CGFobject *obj, char *filnam)`** lädt Grafik-Objekte, die in ASCII-Dateien im hier beispielhaft entwickelten „Computer-Grafik- Format“ (CGF) vorliegen; die Funktion ist unvollständig (ca. 16 Zeilen fehlen).
`main4setCGF.c`	enthält das **`main()`** zur Testung bzw. Nutzung der o.a. Funktionen; die Datei ist „gebrauchsfertig“. Zu Beginn der Übung sollte darin der Makro **`MIT_LOADCGF`** deaktiviert (auskommentiert) sein.

1 Vervollständigung von `setTriangleCGF()`

Das Programm ist im gelieferten Zustand compilierbar, dann aber nicht lauffähig, denn in **`setTriangleCGF()`** wird für das dort definierte Dreieck-Objekt noch kein Speicherplatz reserviert.

Nach Ergänzung von **`setTriangleCGF()`** durch Speicherplatz-Reservierung und entsprechende Belegung der Struktur-Elemente gibt **`main()`** die korrekten Meldungen über die Elemente des Objektes heraus. (Das sind dieselben Angaben wie nach dem Laden eines der bereitgestellten Objekte mit **`loadCGF()`** über die Funktion **`printCGF()`** – s.u.)

Das Dreieck-Objekt soll eine zweite Fläche (Rückseite) bekommen, die später, bei der Darstellung als Grafik, mit einem anderen Symbol als die Vorderseite (**`LGREY`**) dargestellt wird. Die zwei Flächen sollen entgegengesetzte Orientierung haben, d.h., wenn die Punkt-Indizes einer Seite gegen den Uhrzeigersinn geordnet sind, sollen die Punkt-Indizes ihrer Rückseite im Uhrzeigersinn geordnet sein. An dieser Anordnung wird später das Grafik-System erkennen, welche Seite dem Betrachter zu- und welche abgewandt ist.

Um die Effekte kennenzulernen, die mit der einseitigen und der zweiseitigen Definition von Flächen verbunden sind, ist im Code der Makro **DOUBLEFACE** eingerichtet worden; mit ihm können über bedingte Compilierung beide Versionen aktiviert werden.

Abb. 1 zeigt die Ausgaben des Programms nach Fertigstellung des zweiflächigen Objekts:

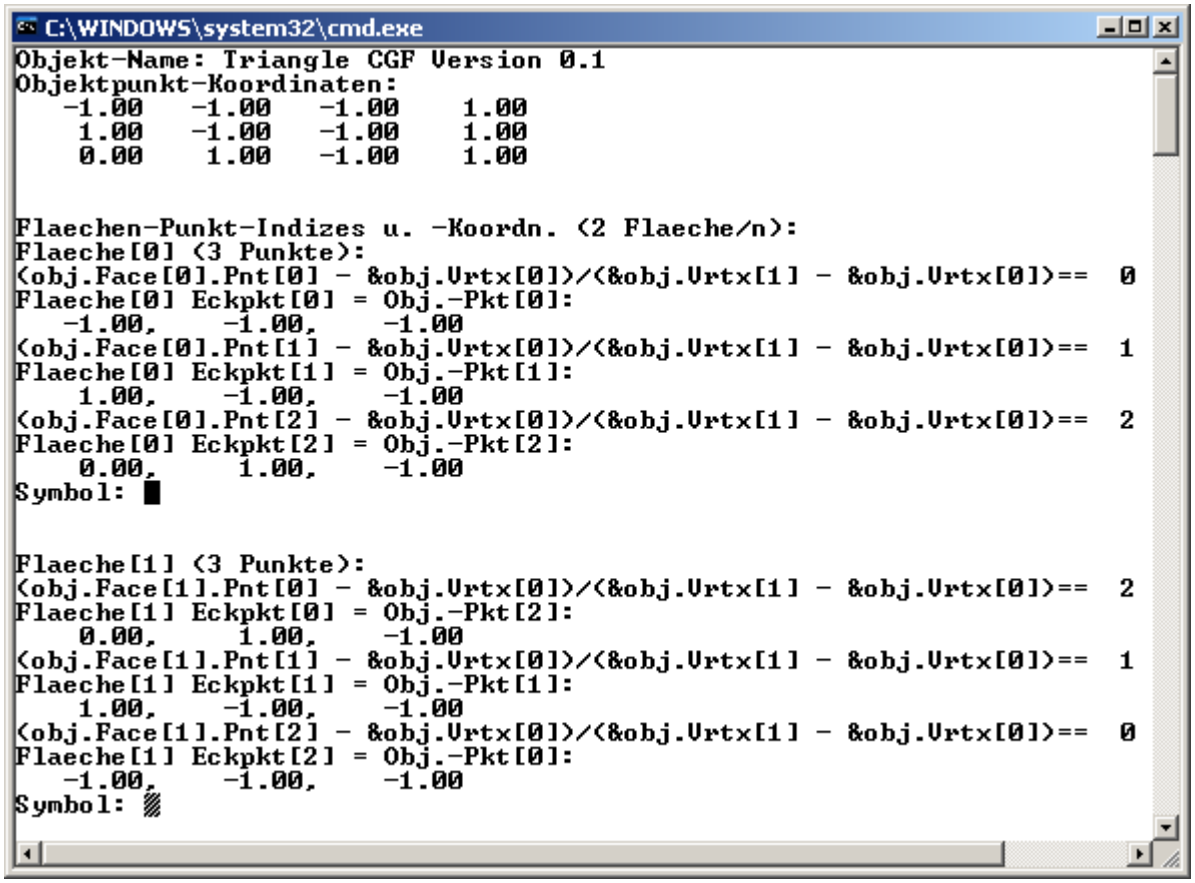

Abb. 1: Meldungen von **`LoadSetCGF`** nach Definition einer zweiten Fläche (Dreieck-Rückseite) und Aufruf von **`setTriangleCGF()`**

2 Vervollständigung von `printCGF()`

Wird im **`main4setCGF.c`** der Bezeichner (Makro) **`MIT_LOADCGF`** aktiviert, so wird zunächst die Adresse des gerade gesetzten Dreiecks der Funktion **`printCGF()`** übergeben; diese dient als Kontrollhilfe für das korrekte Laden grafischer Objekte, indem sie die ausgelesenen Daten alphanumerisch ausgibt. Sie hat exakt die gleiche Struktur wie die vorausgegangene Ausgabe der gleichen Werte in **`main()`** und kann prinzipiell durch Kopieren des entsprechenden Code-Abschnitts (zwischen die vorbereiteten Direktiven mit **`MORE_PRNT`**) eingeleitet werden. Es ist jedoch zu beachten, daß sich hier die Aufgabenstellung leicht geändert hat:

Während in **`main()`** das behandelte Objekt mit seinen Daten verfügbar (weil es dort als Variable definiert) war, soll nun **`printCGF()`** die Daten eines Objektes ausgeben, das typischerweise mit einer anderen Funktion geladen wurde (hier: mit **`loadCGF()`**, nach ihrer Ergänzung im nächsten Abschnitt dieser Übung).

Sinnigerweise wird zu diesem Zweck nicht das ganze (beliebig große) Objekt in die Ausgabe-Funktion kopiert; **`printCGF()`** wird vielmehr die Adresse des Objektes im Arbeitsspeicher mitgeteilt, so daß (ohne unnötigen Aufwand an Rechenzeit und Speicherplatz) diese Aufgabe erfüllt werden kann. Das bedeutet in C, daß der Zugriff auf Teile der Struktur **`CGFobject`** mit dem Pfeil-Operator (`->`) erfolgen muß. Bei Struktur-Elementen, die selbst Strukturen sind, ist entsprechend darauf zu achten, ob sie ihre Elemente als solche (Operator '`.`'), oder als Zeiger (Operator '`->`') enthalten.

3 Vervollständigung von `loadCGF()`

Mit dieser Funktion werden beliebige Grafik-Objekte aus CGF-Dateien geladen. Vor ihrem Aufruf in **`main()`** wird der Name der gewünschten Datei abgefragt. Gibt man keinen existierenden Dateinamen ein, so wird das Würfel-Modell aus der Datei **`cube.cgf`** geladen (Voreinstellung).

Die Lücken im Code sind mit **`MORE_LOAD`** gekennzeichnet; die dort zu ergänzenden Funktionalitäten sind den vorausgeschickten Kommentarzeilen und dem Nutzer-Dialog zu entnehmen (oder im Zweifelsfall mit Hilfe des Autors zu erkunden).

Zur Unterstützung bei diesem Übungsteil wird ein kurzes Repetitorium über den Umgang mit Dateien in C (Anleitung und Projekt) auf derselben Seite der Homepage wie das vorliegende Übungsblatt angeboten.