Übung Nr. 1:

Der Linien-Algorithmus nach Jack E. Bresenham ist für alle Oktanten zu implementieren, und es ist zu überprüfen, ob (bzw. wann) die mit ihm gezogenen Linien abhängig von der Wahl des Anfangs- und Endpunktes sind. Als Pixel sollen ASCII-Zeichen dienen, die im Konsole-Fenster ausgegeben werden (Abb.1).

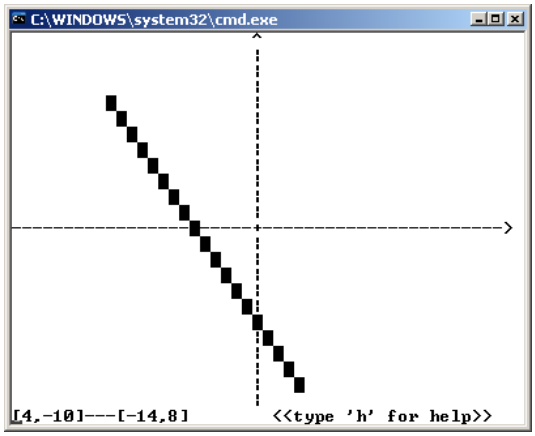


Abb. 1 `BresenLine.exe`

Unter http://homepages.thm.de/christ/ kann ein hierfür vorbereitetes MS-VC-Projekt heruntergeladen werden. Es enthält, neben einem ausführbaren „Funktionsmuster", den codierten Linienalgorithmus in der Version für den 1. Oktanten, die in der Vorlesung behandelt wurde, sowie die benötigte „Infrastruktur" zur Darstellung der erzeugten Linie.

Das gesamte Programm besteht aus drei Quellen- und den zugehörigen Header-Dateien:

`BresenLine.c` `BresenLine.h`	sind „gebrauchsfertig". Sie enthalten die eigentliche „Infrastruktur", die gerade diese konkrete Applikation braucht: die Gestaltung des Fenster-Inhalts, des Bedienungs- und des Hilfe-Menüs, die in dieser Form voraussichtlich nur für diese Implementierung eingesetzt werden. Für den Fall, daß einzelne Funktionen auch in anderen Projekten gebraucht werden, ist für `main()` eine bedingte Compilierung vorgesehen.
`conDraw.c` `conDraw.h`	sind vollständig. Sie stellen jene Funktionen bereit, die das „Pixel-Setzen" (ASCII-Zeichen an beliebigen Stellen) und damit das Zeichnen im Konsole-Fenster ermöglichen. Sie sollen exemplarisch Funktionalitäten vorstellen, die beim erstmaligen Entwurf solcher Systeme unumgänglich sind (vgl. Ersteinführung von Grafik am Telefon, in der Navigation, am Kopier-Automaten, an der Waschmaschine, ...).
`lineOps.c` `lineOps.h`	sollen noch vervollständigt werden. Sie enthalten den C-Code zur (nicht weiter benötigten) Linienziehung mit „roher Gewalt" und zum vorgestellten Bresenham-Algorithmus für den 1. Oktanten; letzterer soll auf die restlichen 7 Oktanten übertragen werden.

Die Übung besteht aus zwei Teilen:

1.1 Erweiterung des Programms auf alle Oktanten

Übertragen Sie den Bresenham-Algorithmus auf die restlichen Oktanten, indem Sie die benötigten Koordinaten an geeigneten Achsen spiegeln, damit der ursprüngliche Algorithmus anwendbar wird. Gewöhnen Sie sich dabei an, modular zu denken, zu konzipieren und zu

programmieren! Vermeiden Sie soweit wie möglich Veränderungen („Aufbohren“) des Originals, fassen Sie wiederkehrende Vorgänge zu Funktionen zusammen[1].

Sorgen Sie bitte dafür (spätestens nach Fertigstellung der ersten Version), daß in Ihrem gesamten Code die Anweisung `conSetPixel()` genau einmal vorkommt!

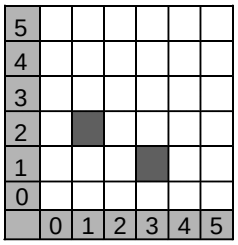
Die Code-Aufteilung im mitgelieferten Code-Fragment sollte auch als ein typisches Beispiel (und Empfehlung) für die Strukturierung größerer Projekte angesehen werden: Funktionen werden nach der Frage der Wiederverwendung unterschieden. (vgl.: „Read / Modify / Write als drei Instruktionen realisieren, oder nur als eine?“) Entsprechend werden Dateien nach der Frage der zusammenhängenden Verwendung getrennt (jeweils alle Menü- / Grafik- / Vektor- / Lade- & Speicher-Funktionen etc.).

Eine wesentliche Rolle spielt dabei die Erkenntnis, daß in den eher seltenen Fällen, in denen eine Codierung die exakte (d.h.: ohne erkennbare Fehler gelungene) Umsetzung der Problemlösung (des sog. Pflichtenhefts) darstellt, bald mit einer Änderung der Aufgabenstellung (des sog. Lastenhefts) zu rechnen ist, mit der neue Leistungsmerkmale (Features) verlangt werden. Die darauffolgende Erstellung des neuen Pflichtenhefts sollte möglichst wenig (Korrektur- / Anpassungs-) Aufwand erfordern.

1.2 Prüfung auf Umkehrbarkeit der Richtung und Anpassung

Testen Sie Ihr Programm nach seiner Fertigstellung und spielen Sie damit, bis Sie den Eindruck haben, daß es wie beabsichtigt läuft.

Versuchen Sie als nächstes, auf dem Papier (auch mit dem Debugger) zu verfolgen, welchen Einfluß die Richtung der Linienziehung hat. Sie können als einfaches Beispiel die Verbindung zwischen den Pixelkoordinaten (1, 2) und (3, 1) betrachten: Welche Pixel werden gesetzt, wenn sich die Richtung umkehrt, von (3, 1) nach (1, 2)?



Überlegen Sie sich, wie Sie den Code anpassen können, so daß er immer über dieselben Koordinaten führt. Es ist Ihnen freigestellt, ob Sie die Änderungen auch codieren. Falls Sie dies tun wollen, sollten Sie dabei bedingte Compilierung verwenden, damit Sie beide Funktionsweisen einstellen können.

Lassen Sie schließlich das Programm sich mit Ihrem Namen melden (berechtigterweise).

...und vergessen Sie nicht, daß bei Bedarf der Autor Ihnen zur Verfügung steht!

[1] Auch bei dieser Aufgabenstellung können acht fast inhaltsgleiche `if`-Anweisungen – oder (besser) drei kurze Funktionen implementiert werden: je eine zur Erkennung des angesprochenen Oktanten, zur Spiegelung der übergebenen Koordinaten und zur (Rück-)Anpassung der beschlossenen Wachstumsrichtung an die ursprünglichen Werte.