Theorie/Praxis – IKA 9/13 - 14.05.2014 / Teil 1

Vorspann: Bitte durchlesen und eingeben!

```
#include "stdafx.h"     // Kennen wir schon...
#include <iostream>     // Kennen wir schon...
#include <windows.h>    // Diese Bibliothek schenkt uns Microsoft...
using namespace std;    // Kennen wir schon...

/* Deshalb brauchen wir Kenntnis von STRUKTUREN und FUNKTIONEN:

  In <windows.h> gibt es eine Struktur namens "COORD",
  diese Struktur dient als Positionsangabe (Koordinaten) eines Cursors auf der Konsole
  mit einem x- und einem y-Wert.

  Außerdem gibt es zwei Funktionen, die den Cursor auf diese Position setzen:
  - GetStdHandle(STD_OUTPUT_HANDLE):
    - STD_OUTPUT_HANDLE ist eine in Windows definierte Konstante,
      eine Art "Gerätenummer" der Art "const int STD_OUTPUT_HANDLE = 3;"
            - die Funktion "GetStdHandle" öffnet uns einen Windows-Prozess
           und gibt uns die Windows-interne "Bearbeitungsnummer" zurück
             <Beispiel> Kfz-Zulassung: da müssen Sie auch eine Nummer ziehen und
           kommen dann dran, wenn die Nummer aufgerufen wird </Beispiel>
  - SetConsoleCursorPosition("Bearbeitungsnummer","gewünschte Cursorposition"):
    Diese Windows-interne Funktion erwartet von uns und deshalb übergeben wir ihr:
            1. die Bearbeitungsnummer (s.o.).
           Die interessiert uns selbst gar nicht und reichen wir deshalb ungeprüft einfach weiter.
            2. Die Cursorposition in x-y-Koordinaten als Datentyp "COORD", siehe oben

  Damit wir nicht jedesmal, wenn wir den Cursor auf eine neue Position setzen wollen, diesen ganzen Sermon
hinschreiben müssen, "kapseln" wir diese Reihe von Anweisungen in eine EIGENE Funktion
(Wiederverwendbarkeit!!!):
  Dieser eigenen Funktion, taufen wir sie einfach "setzeCursor", übergeben wir in unserem Programm nur die
gewünschte Cursor-Position.
  Den Rest - die Abwicklung und Unterhaltung mit Windows und den Bearbeitungsnummern - soll sie gefälligst
selbst bewerkstelligen.
  Wir können uns dann im restlichen Programm auch (fast) nie mehr vertippen - höchstens mal x und y
verwechseln...
  Wir erwarten auch keine Rückmeldung, sondern setzen voraus:
  Wenn die Funktion fertig ist und UNSER übriges Programm wieder dran ist, dann steht der Cursor halt auf der
neuen Position und gut ist!
*/

void setzeCursor(int x, int y) {
     COORD cursor={x,y};
     SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),cursor);
}
```

Probieren Sie es aus und schreiben beliebige Texte an beliebige Positionen auf der Konsole!

Schreiben Sie mit dieser Kenntnis außerdem ein Hauptprogramm, das ein Rechteck an eine beliebige Position auf der Konsole zeichnen kann. Hilfestellung zu den ASCII-Codes:

```
      char Ecke1 = char(218); // Ecke links oben
      char Ecke2 = char(191); // Ecke rechts oben
      char Ecke3 = char(192); // Ecke links unten
      char Ecke4 = char(217); // Ecke rechts unten
      char Strich= char(196); // waagerechter Strich -
      char Linie = char(179); // senkrechter Strich |
```

Theorie/Praxis – IKA 9/13 - 15.05.2014 / Teil 2

Vorspann: Bitte durchlesen und eingeben!

```
#include "stdafx.h"     // Kennen wir schon...
#include <iostream>     // Kennen wir schon...
#include <windows.h>    // Diese Bibliothek schenkt uns Microsoft...
using namespace std;    // Kennen wir schon...
```

In der letzten Übung haben wir uns bereits eine Funktion geschrieben, die uns das Setzen des Cursors an eine beliebige Position auf dem Konsolenbildschirm erleichtert:

```
void setzeCursor(int x, int y) {
     COORD cursor={x,y};
     SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),cursor);
}
```

Aufgabe 1:

Deklarieren Sie eine Struktur „**Rechteck**“ bestehend aus 4 Ganzzahlen für die X-Position, die Y-Position, die Breite und die Höhe des darzustellenden Rechtecks, sowie einem Zeichen für die „Füllung“ des Rechtecks. (char(219) wäre z.B. ein vollständig ausgefülltes Blocksymbol.)

Schreiben Sie eine Funktion „**zeichneRechteck**“, deren Prototyp so aussieht:

```
void zeichneRechteck(Rechteck R);
```

Diese Funktion erhält als Übergabeparameter also ein Rechteck (eigener Datentyp Rechteck, siehe Struktur) und somit auch die Abmessungen und Lage eines Rechtecks. Dieses Rechteck soll nun auf den Bildschirm gezeichnet werden.

Legen Sie im Hauptprogramm ein neues Rechteck **R1** an mit allen benötigten Strukturelementen und rufen Sie die Funktion auf.

Aufgabe 2:

Eine weitere schöne Funktion aus der Windows-Bibliothek **<conio.h>** ist die Funktion:

```
char _getch(); // Prototyp der Funktion in der conio.h-Bibliothek
```

Diese Funktion liefert uns unmittelbar nach den Drücken einer Taste auf dem Keyboard (also ohne die Eingabetaste zu drücken) das auf dieser Taste hinterlegte Zeichen vom Datentyp char.

Damit können wir zum Beispiel die Lage des Rechtecks verschieben: die Tasten **w**, **a**, **s** und **y** könnten das Rechteck entsprechend der Lage der Tasten auf dem Keyboard um 1 Position nach oben, links, rechts oder unten verändern – die Strukturelemente für die Position sind dann entsprechend um 1 zu erniedrigen oder zu erhöhen. Das Drücken der Zifferntaste **0** möge das Programm beenden.

Aufgabe 3:

Programmieren Sie einen Mehrspielermodus, indem Sie ein zweites Rechteck **R2** definieren und z.B. mit den Tasten **i**, **j**, **k** und **m** steuern...

Mögliche Verfeinerungen: Erkennen, ob ein Rechteck überhaupt vollständig auf den Bildschirm passt oder am Rand verschwinden will. Erkennen, ob die beiden Rechtecke kollidieren.