```cpp
// Grundlagen.cpp : Gebrauch von eingebauten Funktionen und eigenen Funktionen
//
#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);
}

int main(int argc, _TCHAR* argv[])
{
    // und hier zum Test:
    setzeCursor(25,25); // void-Funktion gibt nichts zurück = Prozedur = steht ganz links in einem Ausdruck!
    cout << "Test1";    // Wo wird jetzt die Ausgabe der Zeichenkette "Test1" angezeigt? An Position 25,25 !
    setzeCursor(10,15); // Cursor neu setzen...
    cout << "Test2";    // Und wo steht jetzt "Test2" auf dem Bildschirm? An
```

```
    Position 10,15 !
44      int XPos=30, YPos=5;// Hier definieren wir mal 2 Variablen, die wir als
    Koordinate verwenden können
45      setzeCursor(XPos,YPos);
46      cout << "Test3";    // Und nun...
47
48      setzeCursor(0,0);   // ... ganz oben links ...
49      system("pause");
50
51      system("cls");      // Neues Spiel, neues Glück:
52      for (int i=0; i<15; i++) // Hier werden die Koordinaten-Werte "berechnet"
53      {
54          setzeCursor(i,i);
55          cout << '*';
56      }
57      cout << endl;
58      system("pause");
59
60      // Und noch etwas in Richtung "Grafik" (so haben wir in den 1970er Jahren
        angefangen, siehe "Videotext"...
61      system("cls");
62      char Ecke1 = char(218); // Ecke links oben
63      char Ecke2 = char(191); // Ecke rechts oben
64      char Ecke3 = char(192); // Ecke links unten
65      char Ecke4 = char(217); // Ecke rechts unten
66      char Strich= char(196); // waagerechter Strich -
67      char Linie = char(179); // senkrechter Strich |
68
69      // Jetzt erst mal ohne weiteren Schnickschnack ein Rechteck zeichnen:
70      cout << Ecke1 << Strich << Strich << Strich << Ecke2 << endl;
71      cout << Linie << "   " << Linie << endl;
72      cout << Linie << "   " << Linie << endl;
73      cout << Linie << "   " << Linie << endl;
74      cout << Ecke3 << Strich << Strich << Strich << Ecke4 << endl;
75
76      // Jetzt malen wir das Rechteck an eine beliebige Position:
77      XPos = 8;
78      YPos = 8;
79      setzeCursor(XPos,YPos);
80      cout << Ecke1 << Strich << Strich << Strich << Ecke2 << endl;
81      // Jetzt müssen wir eine Zeile tiefer gehen, also YPos um 1 erhöhen, XPos bleibt
         auf 8
82      YPos++;
83      setzeCursor(XPos,YPos);
84      cout << Linie << "   " << Linie << endl;
85      setzeCursor(XPos,++YPos); // Hier gezeigt außerdem: die Präfix-Variante des ++ -
         Operators: erst rechnen dann verwenden.
86      cout << Linie << "   " << Linie << endl;
87      setzeCursor(XPos,++YPos);
88      cout << Linie << "   " << Linie << endl;
89      setzeCursor(XPos,++YPos);
90      cout << Ecke3 << Strich << Strich << Strich << Ecke4 << endl;
91      system("pause");
92
93      // Um ein Rechteck an eine beliebige Position zu schreiben und das dann
        vielleicht noch über den Bildschirm zu bewegen,
```

```
94      // wollen wir ja wohl nicht jedes Mal diese Kette von setzeCursor- und cout-
        Anweisungen hinschreiben, oder?
95      // Was machen wir stattdessen, Stichwort "Wiederverwendbarkeit"?
96      // Richtig: eine Funktion schreiben...
97      // Weiter in Aufgabe1.cpp:
98
99      return 0;
100 }
101
```

```
1  // Aufgabe1.cpp : Definiert den Einstiegspunkt für die Konsolenanwendung.
2  //
3  // Was haben wir bereits aus den Grundlagen?
4  // Wir haben die Funktion "setzeCursor(x,y)" - Wir verwenden sie hier per "Copy&Paste".
5  // (Wäre aber wohl schön, wenn wir die in einer Header-Datei hätten, oder?)
6  // Wir haben auch die Funktionsweise, wie man ein Rechteck auf die Konsole malen kann.
7  // OK: dann kopieren wir das auch hierher und "kapseln" es in einer Funktion, machen das wiederverwendbar.
8  // Die Funktion mit kopiertem Funktionsrumpf folgt hier im Listing weiter unten,
9  // wegen der Forderung "Vor Verwendung einer Funktion muss diese bereits bekannt sein";
10 // und wir verwenden ja die Funktion "setzeCursor", die muss also vorher bereits deklariert sein!!!
11
12 #include "stdafx.h"     // Kennen wir schon...
13 #include <iostream>     // Kennen wir schon...
14 #include <windows.h>    // Diese Bibliothek schenkt uns Microsoft...
15 using namespace std;    // Kennen wir schon...
16
17 // Diese Funktion kennen wir jetzt auch schon, siehe Wuerfeln1:
18 void setzeCursor(int x, int y) {
19      COORD cursor={x,y};
20      SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),cursor);
21 }
22
23 // Hier kommt jetzt unsere neue Funktion:
24 void maleRechteck(int XPos, int YPos) {
25     // Den Funktionsrumpf haben wir 1:1 aus "Wuerfeln1.cpp" kopiert:
26     // Wir nehmen die Wert für "XPos" und "YPos" aber nicht mehr hartkodiert (literal) als "8" und "8" (siehe dort),
27     // sondern wollen die ja variabel als Ausgabgsparameter ("Startbedingung") für unser Funktion verwenden,
28     // aus Sicht unseres Würfelprogramms also als "Übergabeparameter" an unsere Funktion:
29
30     // Ach ja, fast hätten wir es vergessen:
31     // die benutzten Sonderzechen brauchen wir natürlich auch noch.
32     // Weil wir die sonst nicht benutzen, bauen wir sie hier in unsere Funktion ein:
33     char Ecke1 = char(218); // Ecke links oben
34     char Ecke2 = char(191); // Ecke rechts oben
35     char Ecke3 = char(192); // Ecke links unten
36     char Ecke4 = char(217); // Ecke rechts unten
37     char Strich= char(196); // waagerechter Strich -
38     char Linie = char(179); // senkrechter Strich |
39
40     // Aber jetzt kommt die kopierte Funktionsweise:
41     setzeCursor(XPos,YPos);
42     cout << Ecke1 << Strich << Strich << Strich << Ecke2 << endl;
43     // Jetzt müssen wir eine Zeile tiefer gehen, also YPos um 1 erhöhen, XPos bleibt auf 8
44     YPos++;
45     setzeCursor(XPos,YPos);
46     cout << Linie << "   " << Linie << endl;
47     setzeCursor(XPos,++YPos); // Hier gezeigt außerdem: die Präfix-Variante des ++ -
```

```
   Operators: erst rechnen dann verwenden.
48      cout << Linie << "    " << Linie << endl;
49      setzeCursor(XPos,++YPos);
50      cout << Linie << "    " << Linie << endl;
51      setzeCursor(XPos,++YPos);
52      cout << Ecke3 << Strich << Strich << Strich << Ecke4 << endl;
53  }
54
55  int main(int argc, _TCHAR* argv[])
56  {
57      // Wir malen jetzt ein paar Rechtecke an unterschiedlichen Positionen,
58      // zunächst mit von uns festgelegten Werten:
59      maleRechteck(5,15);
60      maleRechteck(20,2);
61      int XPos=25, YPos=7;
62      maleRechteck(XPos,YPos);
63      for (int k=0; k<8; k++)
64          maleRechteck(k,k+1);
65
66      system("pause");
67      system("cls");
68
69      // Weil es so schön ist:
70      // Jetzt pflastern wir den Bildschirm zu mit zufällig erzeugten Rechteck-
        Positionen:
71      // Hoppla: eine Endlosschleife, (Strg-C hilft fürs erste...)
72      while (1==1) {
73          XPos = rand()%70; // linker Rand zufällig an Position 0...69
74          YPos = rand()%20; // oberer Rand zufällig an Position 0...19
75          maleRechteck(XPos,YPos);
76      }
77      system("pause"); // Nutzt hier aber wegen der "Endless Story" nichts mehr...
78      /* Wie geht es weiter?
79      1. Wir könnten nur EIN Rechteck malen und dies per Tastatur oder Maus
        verschieben...
80      2. Wir könnten IN DAS Rechteck HINEIN noch die Augen eines Würfels malen - für
        ein Würfelspiel (Kniffel)
81      3. Wir könnte statt des Rechtecks in der Konsole einen Avatar in HD-Grafik
        bewegen und Kollisionen erkennen...
82      4. Wir machen das alles Schritt für Schritt und haben dafür bis zum Examen Zeit
83      5. wir kommen wieder auf das Thema "Funktionen" und "Strukturen" zurück und
        schauen,
84         wo wir das als IKA-ler noch sinnvoll brauchen können, auch wenn wir keine
           Programmierer-Freaks werden...
85      */
86
87      return 0;
88  }
89
```

```cpp
// Aufgabe1b.cpp : Definiert den Einstiegspunkt für die Konsolenanwendung.
//
#include "stdafx.h"      // Kennen wir schon...
#include <iostream>      // Kennen wir schon...
#include <windows.h>     // Diese Bibliothek schenkt uns Microsoft...
#include <conio.h>       // Stellt kbhit() und getch() zur Verfügung (KeyBordHit: true/false, GetCharacter: Tasteneingabe ohne Return-Taste)
using namespace std;     // Kennen wir schon...

// Diese Funktion kennen wir bereits, siehe Grundlagen.cpp
void setzeCursor(int x, int y) {
     COORD cursor={x,y};
     SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),cursor);
}

// Die Struktur für ein Rechteck, bestehend aus der X/Y-Position,
// den Abmessungen (Breite, Höhe) und einem Symbol für die Füllung
struct Rechteck {
    int XPOS, YPOS, Breite, Hoehe;
    char Symbol;
};

// Der Prototyp der Funktion:
void zeichneRechteck(Rechteck);

int main() {
    // erstmal das Rechteck ohne Werte erzeugen:
    Rechteck R;
    // jetzt die Daten abfragen:
    cout << "Bitte geben Sie die Abmessungen des Rechtecks an:\n";
    cout << "Breite: "; cin >> R.Breite;
    cout << "Hoehe:  "; cin >> R.Hoehe;
    cout << "Bitte geben Sie die Lage des Rechtecks an:\n";
    cout << "X-Position: "; cin >> R.XPOS;
    cout << "Y-Position: "; cin >> R.YPOS;
    // wir könnten auch die Füllung erfragen, setzen sie aber einfach auf:
    R.Symbol = char(219);

    system("cls");  // Bildschirm löschen
    zeichneRechteck(R);
    cout << endl;
    system("pause");
}

// Hier noch die vollständige Funktionsdefinition:
void zeichneRechteck(Rechteck R) {
    for (int i=0; i<R.Hoehe; i++) {
        setzeCursor(R.XPOS,R.YPOS+i);
        for (int j=0; j<R.Breite; j++) {
            cout << R.Symbol;
        }
    }
}
```

```cpp
// Aufgabe2.cpp : Definiert den Einstiegspunkt für die Konsolenanwendung.
//
#include "stdafx.h"      // Kennen wir schon...
#include <iostream>      // Kennen wir schon...
#include <windows.h>     // Diese Bibliothek schenkt uns Microsoft...
#include <conio.h>       // Stellt kbhit() und getch() zur Verfügung (KeyBordHit: true/false, GetCharacter: Tasteneingabe ohne Return-Taste)
using namespace std;     // Kennen wir schon...

// Diese Funktion kennen wir bereits, siehe Grundlagen.cpp
void setzeCursor(int x, int y) {
     COORD cursor={x,y};
     SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),cursor);
}

// Die Struktur für ein Rechteck, bestehend aus der X/Y-Position,
// den Abmessungen (Breite, Höhe) und einem Symbol für die Füllung
struct Rechteck {
    int XPOS, YPOS, Breite, Hoehe;
    char Symbol;
};

// Der Prototyp der Funktion:
void zeichneRechteck(Rechteck);

int main()
{
    char Eingabe;
    Rechteck R2 = {20,20,5,3,219};
    zeichneRechteck(R2);

    do {
        system("cls");
        zeichneRechteck(R2);

        setzeCursor(0,30);
        // cin >> Eingabe;
        Eingabe = _getch();
        switch (Eingabe) {
            case 'a': R2.XPOS--; break;
            case 's': R2.XPOS++; break;
            case 'w': R2.YPOS--; break;
            case 'y': R2.YPOS++; break;
            //default: Eingabe='0';
        }
    } while (Eingabe != '0');

    return 0;
}

void zeichneRechteck(Rechteck R) {
    for (int i=0; i<R.Hoehe; i++) {
        setzeCursor(R.XPOS,R.YPOS+i);
        for (int j=0; j<R.Breite; j++) {
            cout << R.Symbol;
        }
```

```
56      }
57  }
```

```cpp
// Aufgabe2b.cpp : Definiert den Einstiegspunkt für die Konsolenanwendung.
//
#include "stdafx.h"      // Kennen wir schon...
#include <iostream>      // Kennen wir schon...
#include <windows.h>     // Diese Bibliothek schenkt uns Microsoft...
#include <conio.h>       // Stellt kbhit() und getch() zur Verfügung (KeyBordHit: true/false, GetCharacter: Tasteneingabe ohne Return-Taste)
using namespace std;     // Kennen wir schon...

// Diese Funktion kennen wir bereits, siehe Grundlagen.cpp
void setzeCursor(int x, int y) {
     COORD cursor={x,y};
     SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),cursor);
}

// Die Struktur für ein Rechteck, bestehend aus der X/Y-Position,
// den Abmessungen (Breite, Höhe) und einem Symbol für die Füllung
struct Rechteck {
    int XPOS, YPOS, Breite, Hoehe;
    char Symbol;
};

// Der Prototyp der Funktion, um ein Rechteckmzu zeichnen:
void zeichneRechteck(Rechteck);

// weil das system("cls") so blöd flackert,
// probieren wir das mit einer Clear-Funktion:
void clearRechteck(Rechteck);

int main()
{
    char Eingabe;
    Rechteck R2 = {20,20,5,3,219};

    system("cls");
    do {
        zeichneRechteck(R2);

        // setzeCursor(0,30); // Eingabe zum Abbrechen vorbereitet
        // cin >> Eingabe;

        Eingabe = _getch();
        switch (Eingabe) {
            case 'a': clearRechteck(R2); R2.XPOS--; break;
            case 's': clearRechteck(R2); R2.XPOS++; break;
            case 'w': clearRechteck(R2); R2.YPOS--; break;
            case 'y': clearRechteck(R2); R2.YPOS++; break;
            //default: Eingabe='0';
        }
    } while (Eingabe != '0');

    return 0;
}

// Hier die Funktionsdefinitionen:
void zeichneRechteck(Rechteck R) {
```

```
56      for (int i=0; i<R.Hoehe; i++) {
57          setzeCursor(R.XPOS,R.YPOS+i);
58          for (int j=0; j<R.Breite; j++) {
59              cout << R.Symbol;
60          }
61      }
62  }
63
64  void clearRechteck(Rechteck R) {
65      for (int i=0; i<R.Hoehe; i++) {
66          setzeCursor(R.XPOS,R.YPOS+i);
67          for (int j=0; j<R.Breite; j++) {
68              cout << " ";
69          }
70      }
71  }
```

```cpp
// Aufgabe3.cpp : Definiert den Einstiegspunkt für die Konsolenanwendung.
//
#include "stdafx.h"      // Kennen wir schon...
#include <iostream>      // Kennen wir schon...
#include <windows.h>     // Diese Bibliothek schenkt uns Microsoft...
#include <conio.h>       // Stellt kbhit() und getch() zur Verfügung (KeyBordHit: true/false, GetCharacter: Tasteneingabe ohne Return-Taste)
using namespace std;     // Kennen wir schon...

// Diese Funktion kennen wir bereits, siehe Grundlagen.cpp
void setzeCursor(int x, int y) {
     COORD cursor={x,y};
     SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),cursor);
}

// Die Struktur für ein Rechteck, bestehend aus der X/Y-Position,
// den Abmessungen (Breite, Höhe) und einem Symbol für die Füllung
struct Rechteck {
    int XPOS, YPOS, Breite, Hoehe;
    char Symbol;
};

// Der Prototyp der Funktion, um ein Rechteckmzu zeichnen:
void zeichneRechteck(Rechteck);

// weil das system("cls") so blöd flackert,
// probieren wir das mit einer Clear-Funktion:
void clearRechteck(Rechteck);

int main()
{
    char Eingabe;
    Rechteck R1 = {30,30,4,5,176}; // 177, 178
    Rechteck R2 = {20,20,5,3,219};

    system("cls");
    do {
        system("cls");
        zeichneRechteck(R1);
        zeichneRechteck(R2);

        // setzeCursor(0,30); // Eingabe zum Abbrechen vorbereitet
        // cin >> Eingabe;

        Eingabe = _getch();
        switch (Eingabe) {
            // Tasten für das Rechteck R1:
            case 'a': R1.XPOS--; break;
            case 's': R1.XPOS++; break;
            case 'w': R1.YPOS--; break;
            case 'y': R1.YPOS++; break;
            // Tasten für das Rechteck R2:
            case 'j': R2.XPOS--; break;
            case 'k': R2.XPOS++; break;
            case 'i': R2.YPOS--; break;
            case 'm': R2.YPOS++; break;
```

```
56              //default: Eingabe='0';
57          }
58      } while (Eingabe != '0');
59
60      return 0;
61  }
62
63  // Hier die Funktionsdefinitionen:
64  void zeichneRechteck(Rechteck R) {
65      for (int i=0; i<R.Hoehe; i++) {
66          setzeCursor(R.XPOS,R.YPOS+i);
67          for (int j=0; j<R.Breite; j++) {
68              cout << R.Symbol;
69          }
70      }
71  }
72
73  void clearRechteck(Rechteck R) {
74      for (int i=0; i<R.Hoehe; i++) {
75          setzeCursor(R.XPOS,R.YPOS+i);
76          for (int j=0; j<R.Breite; j++) {
77              cout << " ";
78          }
79      }
80  }
```