Übung Nr. 2:

Ein Forscherteam hat ein neuartiges C-Programm implementiert, das unter dem Codenamen „App“ geheimgehalten und ständig verbessert wird. Vom Programm ist vorerst nur soviel bekannt, daß es auf beliebigen Tastendruck ein ganzzahliges Ergebnis liefert und auf mehrfache Nutzung seine Ergebnisse als Zahlenkolonne ausgibt. Bei Betätigung von **<Esc>** terminiert es umgehend.

Sie erhalten den Auftrag, eine „SP-Umgebung“ („SuperProgramming“) zu entwickeln – eine Konsolenanwendung, welche die Ein- und Ausgabe für App übernimmt, ebenso urheberrechtlich geschützt ist und ohne späteren Kontakt zum Forscherteam auch den „Look & Feel“-Erfordernissen der Zeit ständig angepaßt wird.

Sie erkennen, daß, wenn Sie für die Ausgabe zuständig sind, Sie auch eine Funktion brauchen, die den Fenster-Inhalt von App nach einer eventuellen Löschung wiederherstellt. Dazu wird die Einrichtung eines Callback beschlossen.

Vor der Auftragsvergabe werden die Rahmenbedingungen und die Schnittstellen (Aufrufe) für dieses Software-Projekt festgelegt:

Die Forschergruppe erhält von Ihnen

- die Datei SP.h mit notwendigen Prototypen und Makros (für die #include-Anweisungen der Applikationsquellen), darunter mit der Konstanten `SPLINES`; das ist der einzige offengelegte (einsehbare) Teil Ihrer Software.

  Außerdem liefern Sie in compilierter Form (zum Copyright-Schutz):

- Die Initialisierungsfunktion `SPinit()` und die E/A-Funktionen `SPgetch()` und `SPprintf()`, die als Ersatz für die ähnlich lautenden C-Funktionen dienen sollen. Über die Arbeitsweise dieser Funktionen steht fest:
  - `void SPinit (int (*callback)(void));`
    initialisiert Ihr SP-System und teilt ihm die einzige Callback-Funktion von App mit. Erst nach Aufruf von `SPinit()` ist der korrekte Betrieb der „Super-Programming-Umgebung“ gewährleistet.
  - `int SPgetch (void);`
    ist wie `_getch()` zu verwenden.
  - `int SPprintf (char *form, int out);`
    funktioniert wie `printf()` für die Ausgabe genau einer `int`-Variable. Die Funktion erwartet das Format als Zeiger auf eine `char`-Folge von max. `_FORMLEN` Zeichen. Damit sie gleichermaßen für ein- wie (bei Fensterlöschung) für mehrzeilige Ausgaben verwendbar ist, wird vereinbart, daß ihr Aufruf (von der Applikation aus) im Ausgabe-Format keinen Zeilenwechsel enthalten darf – also keine Escape-Sequenzen "`\n`" und "`\r`".

Noch während der Konzeptionsphase Ihres Projektes werden die Rahmenbedingungen geändert: Wird eine der Tasten 'd‘ bzw. 's‘ gedrückt, so soll beim nächsten Aufruf von `SPprintf()` die Zahlenkolonne um je 4 Leerzeichen nach rechts bzw. links innerhalb der Fenstergrenzen der Konsolen-Anwendung verschoben werden (vergleichbar einem beweglichen, rahmenlosen Fenster im Konsolen-Fenster). Sie erkennen sofort, daß diese Forderung nichts an den o.a. Schnittstellen ändert, sondern lediglich Ihre Software-Entwicklung beeinflußt.

Nach Erstellung des ersten Musters erfolgt eine weitere und letzte Änderung, die ebenfalls nur Ihre Software betrifft:

Enthält Ihr SP.h zur Übersetzungszeit die Zeile:

```
#define MIT_TIMER
```

so erhält App nur 30 Sekunden Rechenzeit, unabhängig davon, ob eine Taste gedrückt wird und die Applikation genutzt wird. In der so compilierten Version erscheint immer am rechten Rand der ersten Fensterzeile ein sekundengenauer Countdown.

In beiden Progamm-Versionen haben Sie dafür zu sorgen, daß die bewegliche Zahlenkolonne nicht mit ihren Grenzen (Fenster-Rahmen bzw. Countdown-Ausgabe) kollidiert.

Um auf dem eigenen Rechner alles testen zu können, haben Sie bereits eine fiktive Miniatur der Software Ihres Auftraggebers aufgebaut – nämlich:

- eine scherzhafte Version von App:
  `int calc (int num)`
- eine mögliche beispielhafte Anwendung zur Nutzung Ihrer Software:
  `int example (void)`
- ein Hauptprogramm (zu Testzwecken und deshalb noch mit Nutzung von `printf` und `_getch()` – zumal vor dem Aufruf von `SPinit()`):
  `int main (void)`
- schließlich eine zu Ihrer fiktiven Applikation passende Callback-Funktion, in der Sie schon, wie vereinbart, für die Anzahl der Fenster-Zeilen das Makro `SPLINES` verwenden:
  `int redisplay (void)`

Ihnen ist bewußt, daß vermutlich keine der Funktionen der Original-Version von App denselben Namen hat wie in Ihrer Imitation, aber das beschäftigt Sie nicht weiter. (Warum?)

Aufgabe:

Erstellen Sie die SP-Umgebung nach obiger Beschreibung (Datei SP.c). Die Dateien App.c und App.h mit der o.a. fiktiven Applikation sowie SP.h finden Sie als „Funktionsmuster“ in einem vorbereiteten VC-Projekt.

Berücksichtigen Sie bitte dabei folgende Vorgaben:

- Programmieren Sie Ihre SP-Umgebung so, daß sie nicht nur benutzungs- sondern auch „wartungsfreundlich“ wird: Konzentrieren Sie die Lösung von Unwegsamkeiten (etwa die Verwendung von Zeigern auf Zeiger) auf wenige umgebungsinterne Funktionen (z.B.: `_SPinit`), und halten Sie den Rest Ihrer Software überschaubar!
- Richten Sie „schlanke“ Schnittstellen ein: Setzen Sie keine Variablen in die Parameterliste, deren Präsenz einer Erklärung bedürfte (weil sie nicht wirklich benötigt werden).
- Benutzen Sie möglichst nur (betriebssystem-unabhängige) Anweisungen aus dem Sprachumfang von C und den dazugehörigen Bibliotheken!
- Verwenden Sie in Ihrer SP-Umgebung keine globalen Variablen (spätestens vor Freigabe eliminieren)! Die Eliminierung globaler Variablen in der Applikation bleibt dagegen Hedonist/inn/en vorbehalten.
- Beachten Sie die geforderte strikte Unabhängigkeit zwischen Sw-Umgebung und Applikation! Sowohl die Sw-Umgebung als auch die Applikation sollen jederzeit weiterentwickelt werden können, ohne sich gegenseitig darüber zu unterrichten (bei gleichbleibenden Schnittstellen) – z.B. bei Compilierung von SP.h mit `#define MIT_TIMER`!
- Als „Selbstprüfmuster“ finden Sie ein vorbereitetes Projekt mit weiteren, bereits compilierten Applikationen (App*.obj als Zwischendateien für Visual Studio). Bei korrekter Entwicklung muß auch dieses Projekt mit der Quelle Ihrer SP-Umgebung (mit/ohne Timer) gebunden werden und laufen können (zu Linker-Einstellung s.a. Notizen über die empfohlenen Projekt-Einstellungen). Führen Sie das Ergebnis vor!