C vs. C++ - Übungsaufgaben

1. Klassen definieren, Objekte erzeugen, Text auf der Konsole ausgeben
   a) Besorge dir einen C++-Compiler oder sorge dafür, dass dein Compiler den Code jetzt als C++ interpretiert. Oft reicht es aus, Dateien mit der Endung .cpp (statt .c) zu kompilieren. Importiere nun das Projekt "Bank - Trennung Dek Impl", indem du die Header- und Quelldateien in dein eigenes Projekt importierst. Wenn alles in Ordnung ist, sollte sich das Projekt kompilieren lassen.

   b) In "Konten.h" und "Konten.cpp" befindet sich bereits eine Spezialisierung der Klasse Bankkonto. Nun soll noch eine weitere Spezialisierung hinzukommen. Implementiere die Klasse Sparkonto. Diese Klasse soll von der Klasse Bankkonto erben und ebenfalls eine Funktion "int zahleAus(int wert)" implementieren. Allerdings sollen maximal 50 Geldeinheiten pro Vorgang ausgezahlt werden. Wenn eine Auszahlung von mehr als 50 Geldeinheiten angefordert wird, soll nichts passieren. Trenne die Deklaration von der Implementation. Die Deklaration soll also in "Konten.h" und die Implementation in "Konten.cpp" zu finden sein.

   c) Erzeuge in main() ein Objekt der Klasse Sparkonto und probiere aus, ob deine Implementation funktioniert. Rufe dazu nach jedem Vorgang die Funktion getKontostand() auf und gebe das Ergebnis auf der Konsole aus. Unten findest du bereits eine Vorlage mit den erwarteten Kontoständen. Das Objekt sparkonto1 muss allerdings noch erzeugt werden und die Konsolenausgaben müssen hinzugefügt werden.

```
1 ...
2 cout << "Sparkonto:" << endl;
3 sparkonto1->zahleEin(100); //Kontostand: 100
4 sparkonto1->zahleAus(60); //Kontostand: 100
5 sparkonto1->zahleAus(10); //Kontostand: 90
6 ...
```

   d) Zusatzaufgabe: Von der Klasse Bankkonto soll man in Zukunft keine Objekte erzeugen können. Man muss sich schon für eine konkrete Spezialisierung entscheiden. Außerdem soll immer die spezielle Funktion aufgerufen werden, auch wenn ein Bankkonto-Pointer vorliegt. Löse das Problem mit virtuellen Funktionen und abstrakten Klassen.

2. Importiere die Dateien Main.h und Main.cpp aus dem Projekt "Templates". Ändere das Template der Funktion addiere(), sodass man auch Objekte zweier verschiedener Typen miteinander addieren kann. (z.B. int a und float b)

3. Standardbibliotheken und Kompatibilität
   a) Angenommen du möchtest in C++ malloc() benutzen um für structs Speicher im Heap zu reservieren, da malloc() von der Performance her besser ist als "new". Dafür importierst du aber nicht "stdlib.h", sondern die neuere Version "cstdlib". Du hast nun bereits "cstdlib" importiert (siehe unten). Kannst du jetzt einfach auf die Funktionen von "cstdlib" zugreifen, wie es in C möglich war?

```
1 #include <cstdlib>
```

b) Was musst du beachten, wenn du malloc() benutzt? Ist es so möglich, wie es hier steht?

```
1 ...
2 int main(int argc, char* argv[])
3 {
4     int* a2 = malloc(sizeof(int));
5     return 0;
6 }
```

Zu allen Aufgaben gibt es Musterlösungen im Anhang.