Prof.Dr.H.P.Weber	Praktische Prüfung PAD1 WS 15/16	h_da HOCHSCHULE DARMSTADT UNIVERSITY OF APPLIED SCIENCES fbi FACHBEREICH INFORMATIK

Name: MatNr: User:

Hinweis: Denken Sie während der Klausur auch an die **Datensicherung!** Arbeiten Sie in kleinen Schritten und sorgen Sie dafür, dass Sie jederzeit ein lauffähiges Anwendungssystem haben.

Allgemeine Vorgaben:

- **Benutzen Sie ausschließlich MS Visual Studio Professional 2013.**
- Verwenden Sie möglichst keine globalen Variablen oder Objekte und nur solche, die als **const** deklariert sind.
- Programmieren Sie C++-standardkonform (z.B. keine Verwendung von **<conio.h>**, kein Aufruf von **system**, keine Ein-/Ausgabe im C-Stil!)
- Verwenden Sie die vorgegebenen Bezeichner; zusätzliche eigene Bezeichner sind 'sprechende Bezeichner'.
- Beachten Sie das 'Principle of least privilege' (z.B. 'const-correctness').
- Die öffentliche Schnittstelle der Klassen darf durch **operator<** und **operator==**- Funktionen erweitert werden.
- Alle Variablen/Objekte werden zur Übersetzungszeit erzeugt und direkt mit ihrem Namen angesprochen, d.h. in Ihrem Programm kommt kein **new**-Operator und keine Zeiger-Variable vor.
- Trennung von Klassendefinitionen und –implementierungen in .h- und .cpp-Dateien.

Zur Bewertung:
Insgesamt können Sie 100 Punkte bekommen.
Die Note 'sehr gut' (1,0 und 1,3) erfordert mindestens 93 Punkte; bestanden haben Sie mit mindestens 50 Punkten.
Werden die 'Allgemeinen Vorgaben' nicht eingehalten, gibt es Punktabzug.

Aufgabenstellung:
Schreiben Sie ein Programm, das den Ablauf eines Kartenspiels in verschiedenen Varianten simuliert. Verwenden Sie dazu die Klasse **CardGame** und die unterstützenden Klassen **Player** und **Card** (s. Klassendiagramme):

CardGame
+ nrP: int + deckSize: int - players: array<Player,nrP> - deck: array<Card,deckSize>
+ CardGame() + shuffle(): void + deal(): void + play(bool): void + showPlayers():string + showResult():string

Player
- id: int - sumOfPoints: int - playerDeck: vector<Card>
+ Player(int) + clearPlayerDeck(): void + push_back(Card): void + pop_back(): void + back(): Card + size(): int + clearSumOfPoints(): void + addPoints(int): void + toString(): string + *<erforderliche setter/getter>*

Card
- suit: Suit - face: Face - points: int
+ Card(Suit,Face) + toString(): string + *<erforderliche setter/getter>*

Für die Datenelemente und Elementfunktionen gilt:

- Der Konstruktor von **Card** soll **suit** und **face** mit von außen übernommenen Werten für Kartenfarbe und Kartenwert initialisieren und den Punktwert (**points**) in Abhängigkeit vom Kartenwert setzen. Dabei sollen für
  - den Aufzählungstyp **Suit** für die Kartenfarbe die Werte **{ KARO, HERZ, PIK, KREUZ }**,
  - den Aufzählungstyp **Face** für den Kartenwert die Werte **{ SIEBEN, ACHT, NEUN, ZEHN, BUBE, DAME, KOENIG, ASS }**,
  - für den Punktwert (**points**) die ganzen Zahlen **0** (beim Kartenwert **SIEBEN, ACHT** und **NEUN**), **10** (bei **ZEHN**), **2** (bei **BUBE**), **3** (bei **DAME**), **4** (bei **KOENIG**) und **11** (bei **ASS**)

  verwendet werden.
- **toString** soll die Werte von **suit**, **face** und **points** in einer geeigneten Formatierung als **string** zurückgeben.
- Jeder **Player** hat eine eindeutige Identifikationsnummer **id**, eine bestimmte Anzahl von Karten in einem **vector<Card>** und eine ganze Zahl **sumOfPoints**, die festhält, wie viele Punkte der Spieler zu einem bestimmten Zeitpunkt hat. (Die Punkte ergeben sich durch die Punktwerte der Karten, die der Spieler beim Ausspielen erhält; s. Beschreibung von **play** auf der Rückseite.)
- Der Konstruktor von **Player** setzt **id** auf den von außen als Parameter übergebenen Wert und initialisiert **sumOfPoints**.

- Die Elementfunktion **push_back** von **Player** fügt eine Karte hinten zu **playerDeck** hinzu, **pop_back** entfernt die letzte Karte von **playerDeck** und **back** gibt die letzte Karte zurück (ohne sie zu entfernen).
- Es nehmen 4 Spieler am Spiel teil, d.h. die konstante Klassenvariable **nrP** hat den Wert **4**. Ein Kartenspiel hat 32 Karten, d.h. **deckSize** hat den Wert **32**.
- Der Konstruktor von **CardGame** vergibt eindeutige Identifikationsnummern an die 4 **Player** und erzeugt das **deck** mit den 32 unterschiedlichen Karten mit ihren Werten von **suit**, **face** und **points** (s. **Card**-Konstruktor). Hinweis: Sie können hierzu Schleifen mit passenden **int**-Werten verwenden und die **int**-Werte bei der Übergabe an den **Card**-Konstruktor in **Suit** bzw. **Face** casten.
- **shuffle** mischt das Kartenspiel und **deal** verteilt alle Karten aus dem **deck** reihum an die 4 **Player**. Nachdem alle Karten verteilt sind, werden die **playerDecks** der einzelnen Spieler aufsteigend wie folgt sortiert: Alle **KREUZ**-Karten sind höher als alle **PIK**-Karten, diese sind höher als alle **HERZ**-Karten und diese höher als alle **KARO**-Karten. Innerhalb der einzelnen Farben gilt die Reihenfolge **ASS**, **KOENIG**, **DAME**, **BUBE**, **ZEHN**, **NEUN**, **ACHT** und **SIEBEN** (**ASS** am höchsten und **SIEBEN** am niedrigsten).
- **play** realisiert ein Spiel: In jeder Runde muss jeder Spieler seine höchste Karte ausspielen. Die Punktwerte (**points**) der 4 ausgespielten Karten in jeder Runde werden addiert und die Summe bekommt der Spieler mit der höchsten der 4 ausgespielten Karten in seiner **sumOfPoints** durch Aufruf der Elementfunktion **addPoints** gutgeschrieben. Außerdem werden die in dieser Runde ausgespielten Karten aller Spieler aus den jeweiligen **playerDecks** entfernt. Danach folgt die nächste Runde.
  Nach Abschluss aller Spielrunden sollen die Spieler innerhalb **players** absteigend nach ihren **sumOfPoints** sortiert werden.
  play soll zwei unterschiedliche Arbeitsweisen haben: Entweder werden die **sumOfPoints** der Spieler jeweils nur für ein Spiel (also bei jedem Aufruf von **play** wieder neu) berechnet oder in den **sumOfPoints** der Spieler werden die Ergebnisse für eine ganze Reihe von Spielen aufaddiert. Die gewünschte Arbeitsweise von **play** kann über einen **bool**schen Parameter von außen (s. Anforderungen für **main**) angegeben werden.
- **showPlayers** gibt die Verteilung der Karten auf die 4 Spieler zu einem bestimmten Zeitpunkt in einer geeigneten Formatierung als **string** zurück.
- **showResult** gibt
  - zuerst die Identifikationsnummer des Gewinners des Spiels,
  - dann für alle 4 Spieler Identifikationsnummer und erreichte Punktzahlen
  - und zum Schluss als Kontrolle die Summe der Punktzahlen aller 4 Spieler

  in einer geeigneten Formatierung als **string** zurück.

Das **Anwendungsprogramm (main)** soll die Simulationen durchführen. Dazu sollen der Reihe nach folgende Aktionen ablaufen (es soll also kein Menü realisiert werden):

- Erzeugung eines **CardGame**-Objektes und Aufruf von **deal** und **showPlayers** (ohne zu mischen!).
- Erzeugung eines weiteren **CardGame**-Objektes und Aufruf von **shuffle**, **deal** und **showPlayers**.
  Danach Durchführung eines Spiels mit diesem Objekt durch Aufruf von **play** und Aufruf von **showResult**.
- Erzeugung eines weiteren **CardGame**-Objektes und Durchführung von zehn kompletten Spielen (Mischen, Geben, Ausspielen und Resultat jedes einzelnen Spiels anzeigen) mit diesem Objekt.
- Erzeugung eines weiteren **CardGame**-Objektes und Durchführung von 1000 Spielen (Mischen, Geben, Ausspielen und Gesamtresultat aller 1000 Spiele anzeigen) mit diesem Objekt. Dabei sollen die Punkte, die jeder Spieler in jedem der 1000 Spiele erreicht hat, aufaddiert werden und so die Gesamtpunktzahlen, die sie in allen Spielen erreicht haben, ausgegeben werden (absteigend sortiert nach der Gesamtpunktzahl).

Ergebnis:

Als Ergebnis erwarte ich in jedem Fall ein **ausführbares** Anwendungssystem.
Speichern Sie alle Ihre Projektdateien mit Ausnahme des Debug-Verzeichnisses in Ihrem persönlichen Netzverzeichnis.
Das Anwendungssystem muss sich mit diesen Dateien auf "Knopfdruck" erzeugen lassen.
Löschen Sie Ihr Projekt am Klausurende **nicht** und fahren Sie den Rechner **nicht** herunter.

Lassen Sie das Aufgabenblatt (mit Namen, Matrikel- und Usernummer) an Ihrem Platz liegen.

Bewertung (Punkte):	**maximal**	**erreicht**
Klassendefinitionen	16	
Implementierung **Card**	12	
Implementierung **Player**	12	
Konstruktor **CardGame**	8	
shuffle	6	
deal	10	
play	14	
showPlayers	6	
showResult	6	
main	10	

Gesamtpunkte:	**Note:**