Übung 6

Header & Klassen

A 6.1: Header-Dateien und Präprozessor-Anweisungen (4)

Schreiben Sie zwei kleine C++ - Bibliotheken mathematischer Funktionen.
Dabei soll Bibliothek #1 nur die Funktion `Fakultaet(n)` enthalten, Bibliothek #2 die Funktionen `Binomialkoeffizient(n, k)` und `Summe(n)` (soll die Summe der Zahlen von `1` bis `n` berechnen), wobei die Funktion `Binomialkoeffizient(n, k)` die Funktion `Fakultaet(n)` aus Bibliothek #1 benutzen soll (vgl. A5.2).
Erstellen Sie für jede Bibliothek eine Source- und eine Header-Datei und binden Sie diese in ein C++ - Hauptprogramm ein, in dem Sie zu jeder Funktion ein Beispiel ausgeben, z.B.

```
Fakultaet(5) = 120
Binomialkoeffizient(5, 3) = 10
Summe(10) = 55
```

A 6.2: Objekt "Auto" (8)

Schreiben Sie ein C++ - Programm, in dem Sie eine Klasse `Auto` mit den folgenden Feldern und Methoden definieren und deklarieren:

- Felder: `gefahreneKm`, `maxTankinhalt`, `Benzinstand`, `VerbrauchPro100km`
- Methoden:
  Konstruktor (Standard-, Initialisierungs- und Kopierkonstruktor)
  Zugriffsmethoden für alle Eigenschaften
  `fahre(int km)` Fahre `km` viele Kilometer
  `tanke(int liter)` Tanke `liter` viele Liter Benzin

Beachten Sie dabei die Regeln der OOP (Kapselung !). Beachten Sie weiterhin...

- daß die Methode `fahre(...)` nur dann sinnvoll beendet werden kann, wenn noch genügend Benzin für die Fahrt im Tank ist.
- Daß die Methode `tanke(..)` nur dann sinnvoll beendet werden kann, wenn durch das Tanken der Tank nicht "überläuft".

Geben Sie bei Verletzung dieser Regeln entsprechende Fehlermeldungen aus.

Erzeugen Sie ein Objekt vom Typ `Auto` und "fahren" Sie verschiedene Strecken, wobei mindestens eine der Strecken auf der Fahrt zu einem leeren Tank führen soll. Geben Sie dabei jeweils den Kilometer- und Benzinstand zu Beginn und am Ende der Fahrt aus und tanken Sie zwischendurch mehrmals (wobei ebenfalls mindestens einmal der maximale Tankinhalt überschritten werden soll).

A 6.3: Ein Taschenrechner mit Stack (Teil 1.) (6)

Der *Stack* ist eine der wichtigsten Datenstrukturen in der Informatik. Man kann ihn sich als einen "Stapel" von Elementen eines bestimmten Datentyps vorstellen, wobei Elemente jeweils nur oben auf den Stapel gelegt werden können (`void push (Element e)`) oder von oben vom Stapel genommen werden können (`Element pop( )`). Der *Stack* wird dabei meist als Array von Elementen implementiert, wobei ein Zähler die Position des obersten *Stack*-Elements verwaltet. Weiterhin werden standardmäßig meist noch die Methoden `istLeer()` und `istVoll()` benutzt, die angeben, ob der Stack leer ist, d.h. keine Elemente mehr enthält (und somit z.B. ein `pop( )` unsinnig wäre) oder ob er voll ist, d.h. das Array keine weiteren Elemente mehr aufnehmen kann (und somit ein `push(...)` ebenfalls unsinnig wäre).

(a) Definieren und deklarieren Sie eine Klasse vom Typ *Stack*, die Integerzahlen verwalten kann. Implementieren Sie dazu den Standard-Kontruktor, die Methoden `void push(int e)`, `int pop()`, `istLeer()` und `istVoll()`. Ergänzen Sie die Klasse um eine Methode `gebeAus()`, die den Inhalt des *Stacks* beginnend bei obersten Element ausgibt.
Erzeugen Sie ein Objekt vom Typ *Stack* und führen Sie die folgenden Operationen aus:
```
push(3);  push(4); gebeAus();  pop();  push(7);  gebeAus();
```

Teil 2. folgt auf dem nächsten Übungszettel.