Studienarbeit: Programmierung II, Teil 1

1. Für welchen Datentyp (genauer Typ!) wird im folgenden Beispiel die Template-Funktion `SchleuseDurch` aktiviert? Begründen Sie ausgehend davon den auftretenden Fehler.

```
template<class t>
const t& SchleuseDurch(const t&d)
{
   return d;
}
void main(void)
{
   int& a = SchleuseDurch(3.0);
};
```

2. Sie finden unter den Vorlagen zur Studienarbeit ein Programm zur Implementierung und zum einfachen Test einer Liste (Vorlage zu Aufgabe 4 unter `S:\...\Poesl\ProgrammierungII\Vorlagen\Studienarbeit\stu1\stu1_4`), das aus den drei Einzeldateien `CList.h`, `CList.cpp` und `stu1_4.simpletest.cpp` besteht. Schreiben Sie ein Makefile, das die enthaltenen C++-Dateien (Endung „`.cpp`“) zunächst in Objektdateien (Endung unter Visual Studio: „`.obj`“) übersetzt, die Datei `CList.obj` anschließend in die Bibliothek `CList.lib` stellt und zuletzt `stu1_4.simpletest.obj` und `CList.lib` zu einem ausführbaren Programm (`stu1_4.simpletest.exe`) bindet. Geben Sie die Abhängigkeiten zudem graphisch an.

   Hinweise:

   - Compiler, Librarian und Linker (Binder) haben unter Visual Studio die Programmnamen „`cl`“, „`lib`“ und „`link`“. Sie befinden sich im Installationsverzeichnis des Visual Studio. Der Pfad dorthin ist evtl. vorher entsprechend einzustellen (`setpath`).
   - Als Vorlage ist das Makefile-Beispiel aus der Vorlesung „Programmierung I“ (Abschnitt 8.1: Anwendungen zur Programmentwicklung) sinnvoll.
   - Sehen Sie das Target „`all`“ zum Erzeugen des Gesamtprojekts vor und das Target „`clean`“ zum Löschen aller Dateien, die durch einen Aufruf von „`nmake all`“ erzeugt werden.
   - Achten Sie darauf, dass die jeweiligen Objektdateien auch dann neu zu erzeugen sind, wenn sich die die Header-Datei `CList.h` ändert.

3. Definieren/Implementieren Sie eine Klasse `CKomplex` zum Rechnen mit komplexen Zahlen. Trennen Sie dabei Schnittstellenbeschreibung und Implementierung durch Ablage in eigenen Dateien (`CKomplex.h` und `CKomplex.cpp`).
   a. Definieren Sie Methoden zum Initialisieren (Name: `Init`, Parameter: Real- und Imaginärteil) und zur Ein- und Ausgabe (`Eingabe`/`Ausgabe`, keine Parameter). Sehen Sie für beide Parameter beim Initialisieren Defaultwerte vor.
   b. Definieren Sie eine Methode für die Zuweisung einer komplexen Zahl an eine andere (`SeiGleich`). Sowohl Eingabeparameter als auch Rückgabewert der Methode sollten über Referenzen weitergegeben werden.
   c. Definieren Sie Methoden für Addition und Subtraktion (`Plus`/`Minus`). Die Methoden sollen die zu verknüpfende zweite Zahl jeweils als Eingabe-Referenzparameter nehmen und das Ergebnis als Rückgabewert liefern.

→ bitte wenden

Zu Aufgabe 3:

Hinweise:

- Sie finden auf dem OTH-Dateiserver unter „`S:\...\Poesl\ProgrammierungII\Vorlagen\Studienarbeit\stu1\stu1_3\stu1_3.cpp`“ eine Testanwendung, mit der ihre Klasse (sinnvoll) laufen sollte.
- Die Attribute der Klasse (Real-/Imaginärteil) sollten nicht direkt zugreifbar sein.
- Bei der Implementierung können Sie sich an der Klasse `CRatio` aus der Vorlesung / Übung orientieren. Achten Sie bei „reinen“ Eingabeparametern darauf, dass ihre Werte in den Methoden nicht modifiziert werden können.

4. Implementieren Sie Memberfunktionen für eine Klasse `CList`, die eine Liste, d.h. eine Sammlung von geordneten Elementen (`int`), repräsentiert. Folgende Teile sind schon vorgegeben (Vorlagen „`CList.h`“ und „`CList.cpp`“ im Ordner „`S:\...\Poesl\ProgrammierungII\Vorlagen\Studienarbeit\stu1\stu1_4`“):

   - `Init` dient zur Initialisierung und setzt die Kardinalität auf 0.
   - `ContainsValue` prüft, ob ein bestimmter `int`-Wert in der Liste enthalten ist.
   - `AppendElem` hängt ein Element an die aktuelle Liste an.
   - `RemoveElemAt` entfernt ein Element aus der Liste an der angegebenen Pos.
   - `CopyTo` kopiert die aktuelle Liste in eine andere Liste.

   a. Kodieren Sie eine Methode

      ```
      void CList::Print() const,
      ```

      die die Elemente in der Form `<23, 34, ...>` auf `cout` ausgibt.

   b. Kodieren Sie eine Methode

      ```
      bool CList::Equals(const CList&) const,
      ```

      die zwei Listen auf Gleichheit prüft. Zwei Listen sind gleich, wenn sie exakt die gleichen Elemente in der gleichen Reihenfolge enthalten.

   c. Deklarieren und kodieren Sie eine Methode `CList::Contains(...)`, die zurückliefert (`bool`-Rückgabe!), ob alle Elemente einer als Parameter übergebenen Liste an irgendeiner Position der aktuellen Liste in derselben Reihenfolge enthalten sind.

   d. Deklarieren und kodieren Sie eine Methode `CList::AppendList(...)`, die an die aktuelle Liste die Elemente einer anderen Liste anhängt. Der Status der Ausführung soll mit den vordefinierten `ErrCode`-Rückgabewerten angezeigt werden. Gelingt das Anfügen der anderen Liste nicht, soll die aktuelle Liste unverändert bleiben.

   e. Kodieren Sie eine Methode

      ```
      CList CList::SubList(unsigned offset, unsigned len) const,
      ```

      die den bei `offset` beginnenden Teil der akt. Liste mit `len` Elementen als eigenständige Liste zurückliefert. Achten Sie auf Überschreitungen und schneiden Sie ggf. ab.

   Testen Sie alle Methoden! Entwickeln Sie dazu ein kurzes Testprogramm zur manuellen Eingabe von Listen, wenden Sie ihre Methoden auf diese Listen an und prüfen Sie die Ergebnisse. Legen Sie bitte ihre Eingaben und die Ausgaben ihres Testprogramms der Lösung bei.