Praktikumstermin Nr. 12&13, INF: Dyn. Datenstruktur mittels Klasse: `MyString2`

Dieses Aufgabenblatt umfasst zwei Praktika, da diese thematisch zusammen gehören und eine getrennte Veröffentlichung nachteilig wäre. Insbesondere würde man dann den Zweck der Aufgaben des ersten Praktikums nicht gut nachvollziehen können …

Abgabe der INF-12 oder aller Aufgaben im GIP-INF Praktikum der Woche 10.1.-14.1.2022 (sie können gerne schon alle Aufgaben (INF-12 und INF-13) in der ersten der beiden Wochen abgeben) bzw. der INF-13 Aufgaben im GIP-INF Praktikum der Woche 17.1.-21.1.2022.

2021-12-14: Tutoriumsaufgaben hinzugefügt

2022-01-06: Testfall-Ausgaben betr. Anzahl der „assertions" korrigiert

(Pflicht-) Aufgaben INF-12 & INF-13: Dyn. Datenstruktur mittels Klasse: `MyString2`

In den folgenden Teilaufgaben programmieren Sie schrittweise eine Klasse `MyString2`, deren Objekte sich ähnlich verhalten (was einige ihrer Methoden betrifft) wie die Objekte der Klasse `string` aus der C++ Standardlibrary.

Nur die interne Datenspeicherung ist bei der Klasse `string` anders (und viel effizienter) als bei ihrer Klasse, die jeden Buchstaben des `MyString2` Objekts einzeln speichern wird mittels "Knoten" Datenstrukturen auf dem Heap, so wie wir das schon von der einfach verketteten Liste kennen … Diesmal programmieren Sie die "Knoten" Datenstruktur aber als Klasse, nicht als `struct` …

(Pflicht-) Teil-Aufgaben INF-12.01: *Buchstaben-Knoten* für eine *interne einfach verkettete Liste* als Klasse: `CharListenKnoten`

Erstellen Sie in einem neuen Projekt eine leere Headerdatei `catch.h` und kopieren Sie den Inhalt aus der entsprechenden Datei in Ilias in diese leere Datei.

Erstellen Sie eine Datei `main.cpp` mit folgendem Inhalt (Datei auch in Ilias verfügbar):

```
// Datei: main.cpp

#include <iostream>

#define CATCH_CONFIG_RUNNER
#include "catch.h"

int main()
{
    Catch::Session().run();

    system("PAUSE");
    return 0;
}
```

Programmieren Sie (in einer Headerdatei `CharListenKnoten.h` und einer Datei `CharListenKnoten.cpp`) eine **Klasse** `CharListenKnoten` ähnlich zur *`struct`* `TListenKnoten` des Vorlesungsthemas *Dynamische Datenstrukturen*, aber halt als „richtige Klasse" `class CharListenKnoten { … };`

- Die als Attribut realisierte „Nutzlast" jedes `CharListenKnoten` sei ein `char data`.
- Die `CharListenKnoten` seien einfach verkettet in Vorwärtsrichtung über einen `CharListenKnoten* next`.
- Die beiden Attribute seien gegen den Zugriff von außen und aus abgeleiteten Klassen (auch wenn es in diesem Praktikum keine geben wird) geschützt. Programmieren Sie die Setter und Getter für die beiden Attribute.
- Der (einzige) Konstruktor der Klasse nehme zwei Parameter:
  Der erste Parameter sei ein `char` Wert. Der Konstruktor soll dann das `data` Attribut des Objekts mit diesem Wert initialisieren. Für diesen Parameter soll es keinen Default-Wert geben.
  Der zweite Parameter sei vom Typ `CharListenKnoten*` und habe den Defaultwert `nullptr`. Mit diesem Parameterwert soll das Attribut `next` des Objekts initialisiert werden.

Legen Sie die Datei `test_charlistenknoten.cpp` an (Datei in Ilias verfügbar).

Testen Sie ihre Klasse `CharListenKnoten` durch Ausführen dieser Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (6 assertions in 2 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-12.02: Eindeutige ID:s für die `CharListenKnoten` Objekte

Erweitern Sie ihre Klasse `CharListenKnoten` um ein statisches Klassenattribut `next_available_id` vom Typ `int`. Dieses Klassenattribut soll von außerhalb zugreifbar sein.

Initialisieren Sie in der Datei `CharListenKnoten.cpp` das statische Klassenattribut mit dem Wert `1`.

Erweitern Sie ihre Klassendefinition von `CharListenKnoten` so, dass jedes Objekt ein (neues, zusätzliches) `int` Attribut `my_id` besitzt. Bei jedem neuen Anlegen eines `CharListenKnoten` Objekts soll das `my_id` Attribut des Objekts mit dem aktuellen Wert von `next_available_id` initialisiert werden und der Wert von `next_available_id` danach um `1` erhöht werden.
Das `my_id` Attribut sei gegen den Zugriff von außen und aus abgeleiteten Klassen (auch wenn es in diesem Praktikum keine geben wird) geschützt. Programmieren Sie den Getter `get_my_id()` für dieses Attribut. Einen Setter brauchen Sie nicht zu programmieren, da es nicht möglich sein soll, die ID eines Objekts von außen zu setzen bzw. zu ändern.

Das Löschen von Objekten habe keine Auswirkungen auf `next_available_id`. Sie brauchen also den Destruktor (erst einmal) nicht zu programmieren.

Legen Sie die Datei `test_charlistenknoten_id.cpp` an (Datei in Ilias verfügbar).

Testen Sie ihre Klasse `CharListenKnoten` durch Ausführen der Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (9 assertions in 3 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-12.03: Zähler für die `CharListenKnoten` Objekte

Erweitern Sie ihre Klasse `CharListenKnoten` um ein statisches Klassenattribut `object_count` vom Typ `int`. Dieses Klassenattribut soll von außerhalb zugreifbar sein.

Erweitern Sie die Datei `CharListenKnoten.cpp`: Initialisieren Sie in dieser Datei das statische Klassenattribut mit dem Wert `0`.

Erweitern Sie ihre Klassendefinition von `CharListenKnoten` so, dass bei jedem Anlegen eines `CharListenKnoten` Objekts der `object_count` um `1` erhöht wird. Da dieses statische Klassenattribut ja schon existiert und nicht mit dem Objekt angelegt und initialisiert wird, können Sie diese Wertänderung in den Rumpf des Konstruktors programmieren.

Programmieren Sie den Destruktor für `CharListenKnoten`, so dass bei jedem Löschen eines `CharListenKnoten` Objekts der `object_count` um `1` verringert wird.

Legen Sie die Datei `test_charlistenknoten_count.cpp` an (Datei in Ilias verfügbar).

Testen Sie ihre Klasse `CharListenKnoten` durch Ausführen der Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (11 assertions in 4 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-12.04: Funktion `hinten_anfuegen()`

Erweitern Sie die Datei `CharListenKnoten.cpp` um eine Funktion (keine Methode der Klasse, sondern eine normale Funktion) ...

```
void hinten_anfuegen(CharListenKnoten*& anker,
                     const char wert)
```

... sehr ähnlich der entsprechenden Funktion für die einfach verkettete Liste. Sie werden den Code leicht anpassen müssen, da der Konstruktor der Klasse `CharListenKnoten` für den neuen Knoten mindestens einen Parameter erwartet und da Sie das Attribut `next` nur über den Getter und Setter erreichen können.

Fügen Sie den Prototypen der Funktion auch der Datei `CharListenKnoten.h` hinzu.

Neue Datei `test_hinten_anfuegen.cpp`, in Ilias verfügbar.

Testen Sie ihre Klasse `CharListenKnoten` durch Ausführen der Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (23 assertions in 5 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-12.05: Funktion `loesche_alle()`

Erweitern Sie die Datei `CharListenKnoten.cpp` um eine Funktion (keine Methode der Klasse, sondern eine normale Funktion) ...

```
void loesche_alle(CharListenKnoten*& anker)
```

... welche die alle miteinander verketteten `CharListenKnoten` Objekte löscht, auf deren erstes Objekt der `anker` zeigt. Der `anker` soll von der Funktion auf den `nullptr` Wert gesetzt werden.
Sollte der `anker` den `nullptr` als Wert haben, so soll die Funktion sofort zurückspringen und nichts löschen.

Fügen Sie den Prototypen der Funktion auch der Datei `CharListenKnoten.h` hinzu.

Neue Datei `test_loesche_alle.cpp`, in Ilias verfügbar.

Testen Sie ihre Klasse `CharListenKnoten` durch Ausführen der Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (28 assertions in 7 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-12.06: Funktion `deep_copy()`

Erweitern Sie die Datei `CharListenKnoten.cpp` um eine Funktion (keine Methode der Klasse, sondern eine normale Funktion) ...
`CharListenKnoten* deep_copy(CharListenKnoten* orig)`
... welche eine tiefe Kopie der miteinander verketteten `CharListenKnoten` Objekte erstellt, auf deren erstes Objekt der Pointer `orig` zeigt.
Tiefe Kopie bedeutet, dass zu jedem Knoten der Parameter-Verkettung ein neuer, kopierter Knoten erstellt wird (und nicht nur zum ersten Knoten).
Sollte der Parameter `orig` den `nullptr` als Wert haben ("leere Kette von Ursprungsknoten"), so soll auch der `nullptr` als Resultatwert zurückgegeben werden.

Fügen Sie den Prototypen der Funktion auch der Datei `CharListenKnoten.h` hinzu.

Neue Datei `test_deep_copy.cpp`, in Ilias verfügbar.

Testen Sie ihre Klasse `CharListenKnoten` durch Ausführen der Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (40 assertions in 10 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-13.01: Klasse `MyString2`

Programmieren Sie in einer neuen Headerdatei `MyString2.h` und einer neuen Datei `MyString2.cpp` eine Klasse `MyString2`, welche nach außen Funktionalität analog zur C++ Standardklasse `std::string` anbietet (was genau gefordert ist, wird später in dieser Aufgabenstellung noch detailliert dargestellt).

Die Klasse speichere die Buchstaben der Zeichenkette intern dadurch, dass die Buchstaben in einer dynamischen Liste von `CharListenKnoten` auf dem Heap gespeichert werden, jeder Buchstabe einzeln in einem eigenen `CharListenKnoten`.
Es sollen auch leere Zeichenketten gespeichert werden können.
Es sollen nie mehr `CharListenKnoten` vorhanden sein als auch wirklich Buchstaben zu speichern sind.
Es gebe auch keine interne Nullterminierung im `MyString2`, d.h. es werden nur die „wirklichen Buchstaben" als `CharListenKnoten` Einträge gespeichert.

Die Klasse `MyString2` habe dazu nur ein von außen und in abgeleiteten Klassen nicht sichtbares Attribut `CharListenKnoten* anker`, welches den Pointer auf den ersten `CharListenKnoten` (falls vorhanden, sonst `nullptr`) speichert.
Programmieren Sie auch einen Getter `get_anker()` und einen Setter `set_anker()` für dieses Attribut.
Programmieren Sie ferner einen Standard-Konstruktor, der `anker` mit dem `nullptr` Wert belegt.

Neue Datei `test_mystring2_step_1.cpp`, in Ilias verfügbar.

Testen Sie ihre Klasse `MyString2` durch Ausführen der Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (43 assertions in 11 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-13.02: Destruktor für die Klasse `MyString2`

Ergänzen Sie ihre Klasse `MyString2` um einen Destruktor, da die `CharListenKnoten` auf dem Heap interne Ressourcen der `MyString2` Objekte darstellen.

Der Destruktor soll die `loesche_alle()` Funktion nutzen.

Neue Datei `test_mystring2_destruktor.cpp`, in Ilias verfügbar.

Testen Sie ihre Klasse `MyString2` durch Ausführen der Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (45 assertions in 12 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-13.03: Weiterer Konstruktor für die Klasse `MyString2`

Ergänzen Sie ihre Klasse `MyString2` um einen weiteren Konstruktor, der das neue `MyString2` Objekt aus einem `std::string` (also aus einem `string` der C++ Standardlibrary) initialisiert.

Nutzen Sie ggfs. die Funktion `hinten_anfuegen()` geeignet.

Neue Datei `test_mystring2_from_string.cpp`, in Ilias verfügbar.

Testen Sie ihre Klasse `MyString2` durch Ausführen der Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (52 assertions in 13 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-13.04: Copy-Konstruktor und Assignment-Operator für die Klasse `MyString2`

Ergänzen Sie ihre Klasse `MyString2` um einen Copy-Konstruktor und einen Assignment Operator, da mit den `CharListenKnoten` auf dem Heap interne Ressourcen zu verwalten sind.

Nutzen Sie die Funktionen `deep_copy()` und `loesche_alle()` geeignet.

Neue Datei `test_mystring2_copy_assignment.cpp`, in Ilias verfügbar.

Testen Sie ihre Klasse `MyString2` durch Ausführen der Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (80 assertions in 15 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-13.05: Methode `length()` für die Klasse `MyString2`

Ergänzen Sie ihre Klasse `MyString2` um eine Methode …
`unsigned int MyString2::length() const`
… welche analog zur entsprechenden Methode von `std::string` die Länge des `MyString2` zurückgibt.

Neue Datei `test_mystring2_length.cpp`, in Ilias verfügbar.

Testen Sie ihre Klasse `MyString2` durch Ausführen der Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (82 assertions in 16 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-13.06: Methode `at()` für die Klasse `MyString2`

Ergänzen Sie ihre Klasse `MyString2` um eine Methode …
`char MyString2::at(unsigned int pos) const`
… zum Lesen des Buchstabens an der angegebenen Position.

Falls der `MyString2` an dieser Position (Zählung ab Null) gar keinen Buchstaben hat, soll `'\0'` als Ergebnis zurückgegeben werden.
Die Methode `at()` von `std::string` kann man auch auf der linken Seite einer Wertzuweisung benutzen, d.h. man kann mittels der Methode auch Buchstaben im `std::string` ändern; dies braucht ihre Methode nicht zu können.

Neue Datei `test_mystring2_at.cpp`, in Ilias verfügbar.

Testen Sie ihre Klasse `MyString2` durch Ausführen der Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (91 assertions in 17 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-13.07: Methode `to_string()` für die Klasse `MyString2`

Ergänzen Sie ihre Klasse `MyString2` um eine Methode …
`std::string MyString2::to_string() const`
… die einen `std::string` zurückliefert, der eine Kopie der Buchstaben des `MyString2 *this` enthält.

Neue Datei `test_mystring2_to_string.cpp`, in Ilias verfügbar.

Testen Sie ihre Klasse `MyString2` durch Ausführen der Unit Tests.

(Nicht vorzuzeigender) Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (93 assertions in 18 test cases)

Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgaben INF-13.08: Operator-Methode `operator+()` für die Klasse `MyString2`

Ergänzen Sie ihre Klasse `MyString2` um eine Methode …
`MyString2 MyString2::operator+(char c) const`
… mit der man einen Buchstaben an einen `MyString2` anhängen kann und einen *neuen* `MyString2` als Resultat bekommt.
Das aktuelle Objekt `*this` soll dabei nicht verändert werden, d.h. es soll sich um eine konstante Methode handeln.

Neue Datei `test_mystring2_operator_plus.cpp`, in Ilias verfügbar.

Testen Sie ihre Klasse `MyString2` durch Ausführen der Unit Tests.

Vorzuzeigender Gesamt-Testlauf *(keine Benutzereingaben)*:

```
===============================================================
All tests passed (106 assertions in 19 test cases)

Drücken Sie eine beliebige Taste . . .
```

Für das Aufgaben-Tutorium am Freitag 14.1.2022, als freiwillige Aufgaben:

Ein Versandservice versendet Briefe und Pakete und identifiziert diese über eine ID. Zusätzlich werden der Adressat und der Absender gespeichert, sowie ein Flag, das anzeigt, ob das Versandstück bereits zugestellt wurde. Dazu dient eine Klasse Mail, die die gemeinsamen Eigenschaften eines Briefes und eines Pakets darstellt.

(Freiwillige) Aufgabe INF-12.07: UML Klassendiagramm & Programmierung der C++ Klasse Address

Jeder Brief und jedes Paket »hat« zwei Adressen. Definieren Sie daher zunächst eine Klasse Address, die folgende Adressangaben verwaltet:

- Nachname Vorname
- Straße Hausnummer
- PLZ Stadt
- Land

Entsprechend besitzt die Klasse vier Datenelemente vom Typ string: name, street, city, country. Diese seien sowohl von außerhalb als auch aus abgeleiteten Klassen nicht zugreifbar. Definieren Sie Setter und Getter für diese Attribute.

Stellen Sie zur Initialisierung außer einem Standard-Konstruktor, der eine leere Adresse (nur leere Stringwerte in den Attributen) erzeugt, auch einen Konstruktor mit vier Parametern bereit. Der vierte Parameter soll für das Land einen Defaultwert, z.B. "Deutschland", besitzen.

Eine Adresse mit einem leeren String — mit Ausnahme des Landes - ist ungültig. Ob eine Adresse gültig oder ungültig ist, soll mit der Methode is_valid() abgefragt werden können.

Stellen Sie zum Einlesen einer Adresse von der Tastatur die Methode scan() bereit, die keinen Parameter besitzt. Wird beim Einlesen eines Werts — mit Ausnahme des Landes - nur die Return-Taste gedrückt, ist die Eingabe ungültig und die Methode liefert false zurück, andernfalls true. Wird bei Eingabe des Landes nur die Return-Taste gedrückt, so soll das Land "Deutschland" in das Objekt eingetragen werden.
Bei einer ungültigen Eingabe soll das Objekt gar nicht geändert werden.

Überladen Sie die globale Funktion für den Ausgabe-Operator <<, um die Klasse in formatierter Ausgabe auf den Bildschirm (vgl. Testlauf) ausgeben zu können. Deklarieren Sie die Operatorfunktion zum friend der Klasse, so dass im Rumpf der Operatorfunktion auf die Attribute des Address Objekts zugegriffen werden kann.

a) Erstellen Sie das UML-Klassendiagramm der Klasse Address.
Hinweis: Die globale Operatorfunktion wird nicht im UML Diagramm berücksichtigt, da sie nicht Teil der Klasse ist.

b) Programmieren Sie in einer vollständigen Headerdatei Address.h den C++ Code der Klasse Address. Nur die Methode scan() sowie die globale Operator-Funktion sollen in der .cpp-Datei Address.cpp programmiert werden.

(Freiwillige) Teil-Aufgabe INF-12.08: UML Klassendiagramm & Programmierung der C++ Klasse Mail

Entwerfen Sie nun eine Klasse Mail mit Attributen id vom Typ int für die ID, from und to zum Speichern der Adressen sowie einem Flag delivered, das den Wert true aufweist, falls die Post ausgeliefert wurde. Die Attribute sollen aus abgeleiteten Klassen zugreifbar sein, von außen aber nicht. Definieren Sie Setter und Getter für diese Attribute. Eine neue Adresse soll aber nur gesetzt werden können, wenn sie gültig ist.

Definieren Sie einen Konstruktor, der als Argument die ID des neuen Mail-Objekts erhält, sowie einen zweiten Konstruktor, dem zusätzlich noch die Adressen des Absenders und des Empfängers übergeben werden.

Ferner soll ein Versandstück soll mittels der Methde deliver() ausgeliefert werden (Attribut delivered werde in der Methode auf true gesetzt), aber nur dann, wenn die Empfängeradresse gültig ist (dann true als Rückgabewert von deliver(), sonst false).

Definieren Sie außerdem die Methoden scan_from() und scan_to() zum Einlesen des Senders und Empfängers von der Tastatur. Die Rückgabewerte dieser Methoden sollen dieselben sein wie die der entsprechenden Methode in der Klasse Address.

Definieren Sie auch eine polymorph aufrufbare Methode delivery_text(), die abhängig vom Wert des delivered Flags den String "Ausgeliefert an *<Empfängername>*" (mit dem spezifischen Empfängernamen) oder "Noch nicht ausgeliefert!" zurückgibt.

Überladen Sie auch für die Klasse Mail die globale Operator-Funktion <<. Diesmal soll die Operator-Funktion nicht als friend deklariert werden.

a) Erstellen Sie das UML-Klassendiagramm der Klasse Mail.
Hinweis: Die globale Operatorfunktion wird nicht im UML Diagramm berücksichtigt, da sie nicht Teil der Klasse ist.

b) Programmieren Sie in einer vollständigen Headerdatei Mail.h den C++ Code der Klasse Mail. Alle Methoden (außer den Konstruktoren)

sowie die globale Operator-Funktion sollen in der .cpp-Datei Mail.cpp programmiert werden.

(Freiwillige) Teil-Aufgabe INF-12.09: Hauptprogramm

Nutzen Sie das folgende Hauptprogramm:

```
// Datei: main.cpp

#include <iostream>
using namespace std;

#include "Mail.h"

int main()
{
    Mail aMail(1020304);

    cout << "Bitte Absender angeben:" << endl;
    if (!aMail.scan_from())
        cout << "Ungueltige Eingabe!" << endl << endl;

    cout << "und der Empfaenger:" << endl;
    if (!aMail.scan_to())
        cout << "Ungueltige Eingabe!" << endl;

    cout << aMail << endl;
    cout << "... Post mit der ID " << aMail.get_id();
    if (aMail.deliver())
        cout << " wurde an " << aMail.get_to().get_name()
        << " ausgeliefert!" << endl;
    else
        cout << " konnte nicht ausgeliefert werden!" << endl;

    system("PAUSE");
    return 0;
}
```

Testlauf (Benutzereingaben sind unterstrichen):

```
Bitte Absender angeben:
Nachname, Vorname: Mustermann, Max
Strasse mit Hausnr.: Turmstr. 12
Postleitzahl Stadt: 12345 Aachen
Land:
und der Empfaenger:
Nachname, Vorname: Mustermann, Maria
Strasse mit Hausnr.: Badstr. 34
Postleitzahl Stadt: 67890 Aachen
```

```
Land:
Mail-ID: 1020304
Absender:
Mustermann, Max
Turmstr. 12
12345 Aachen
Deutschland
Empfaenger:
Mustermann, Maria
Badstr. 34
67890 Aachen
Deutschland
Post wurde noch nicht ausgeliefert!
... Post mit der ID 1020304 wurde an Mustermann, Maria ausgeliefert!
Drücken Sie eine beliebige Taste . . .
```

(Freiwillige) Teil-Aufgabe INF-12.10: Vererbung: UML Klassendiagramm & Programmierung der C++ Klasse Letter

Programmieren Sie eine Klasse Letter, die von der Klasse Mail erbt.

Ein Brief kann als Standard (STANDARD)- oder Eilbrief (EXPRESS) verschickt werden. Definieren Sie deshalb in der Headerdatei Letter.h, aber außerhalb der Klassendefinition, einen Aufzählungstyp Category mit den alternativen Werten STANDARD und EXPRESS.

Die Klasse Letter stelle wie die Basisklasse Mail zwei Konstruktoren bereit, welche die Parameter der entsprechenden Konstruktoren von Mail besitzen und zusätzlich einen weiteren Parameter für die Kategorie des Briefes, mit dem Default-Wert STANDARD.

Die Objekte der Klasse Letter sollen ein Attribut ctg des Aufzählungstyps Category besitzen, welches weder aus abgeleiteten Klassen noch von außen zugreifbar sein soll. Definieren Sie den Setter und Getter für dieses Attribut.

Spezialisieren Sie die Basisklassen-Methode delivery_text() in der Klasse Letter (unter Nutzung der Basisklassenmethode), indem dort der String "Brief: " dem von der Basisklassen-Methode ausgegebenen Text vorangestellt wird.

Überladen Sie auch für die Klasse Letter die globale Operator-Funktion <<, ohne diese als friend zu deklarieren. Ausgabe siehe Testlauf: Zeile „--- Brief-Daten ---“ gefolgt von Zeile mit der Kategorie gefolgt von den Ausgaben für Mail: Mail-ID, Absender, Empfänger, delivery_text.

a) Erstellen Sie das UML-Klassendiagramm für die Klasse Mail und ihre abgeleitete Klasse Letter.

b) Programmieren Sie in einer vollständigen Headerdatei Letter.h den C++ Code der Klasse Letter. Nur die Methode print() sowie die globale Operator-Funktion sollen in der .cpp-Datei Letter.cpp programmiert werden.

Nutzen Sie für den Testlauf das folgende Hauptprogramm:

```
// Datei: main.cpp

#include <iostream>
using namespace std;

#include "Address.h"
#include "Mail.h"
#include "Letter.h"

int main()
{
    // --- Briefe ---
    Address to("Boss, Anton", "Antonstr. 11", "23456 Hamburg");

    // Zwei Briefe ...
    Letter letter1(102030, EXPRESS),
        letter2(203040, Address{}, to, STANDARD);

    cout << "Die Briefe:" << endl << letter1 << letter2 << endl;

    cout << "Bitte Absender fuer Brief " << letter1.get_id()
         << " eingeben:" << endl;
    if (!letter1.scan_from())
        cout << "Ungueltige Eingabe!" << endl;

    cout << "und der Empfaenger:" << endl;
    if (!letter1.scan_to())
        cout << "Ungueltige Eingabe!" << endl;

    cout << "Die neuen Daten des Briefs: \n" << letter1 << endl;

    cout << "Ein Versuch, den Brief auszuliefern:" << endl;
    if (letter1.deliver())
        cout << "Brief mit der ID " << letter1.get_id()
             << " wurde ausgeliefert!\n" << endl;
    else
        cout << "Kein gueltiger Empfaenger!" << endl;

    system("PAUSE");
    return 0;
```

```
}
```

Testlauf (Benutzereingaben sind unterstrichen):

```
Die Briefe:
--- Brief-Daten ---
Kategorie: Express
Mail-ID: 102030
Kein gueltiger Absender!
Kein gueltiger Empfaenger!
Post wurde noch nicht ausgeliefert!
--- Brief-Daten ---
Kategorie: Standard
Mail-ID: 203040
Kein gueltiger Absender!
Empfaenger:
Boss, Anton
Antonstr. 11
23456 Hamburg
Deutschland
Post wurde noch nicht ausgeliefert!

Bitte Absender fuer Brief 102030 eingeben:
Nachname, Vorname: Mustermann, Max
Strasse mit Hausnr.: Turmstr. 12
Postleitzahl Stadt: 12345 Aachen
Land: (leere Eingabe, d.h. ENTER gedrückt)
und der Empfaenger:
Nachname, Vorname: Mustermann, Maria
Strasse mit Hausnr.: Badstr. 34
Postleitzahl Stadt: 67890 Aachen
Land: (leere Eingabe, d.h. ENTER gedrückt)
Die neuen Daten des Briefs:
--- Brief-Daten ---
Kategorie: Express
Mail-ID: 102030
Absender:
Mustermann, Max
Turmstr. 12
12345
Deutschland
Empfaenger:
Mustermann, Maria
Badstr. 34
67890 Aachen
Deutschland
Post wurde noch nicht ausgeliefert!
```

```
Ein Versuch, den Brief auszuliefern:
Brief mit der ID 102030 wurde ausgeliefert!

Drücken Sie eine beliebige Taste . . .
```

(Freiwillige) Teil-Aufgabe INF-12.11: UML Klassendiagramm & Programmierung der C++ abgeleiteten Klasse Parcel

Programmieren Sie eine weitere Klasse Parcel, die auch von der Klasse Mail erbt.

Objekte der Klasse Parcel sollen zwei von außen und aus abgeleiteten Klassen nicht sichtbare Attribute besitzen. Ein Attribut weight vom Typ double für das Gewicht besitzen sowie ein Attribut insured für den Versicherungsstatus vom Typ bool mit Default-Wert false.
Für beide Attribute sollen Getter & Setter programmiert werden.

Die Klasse Parcel stelle wie die Basisklasse Mail zwei Konstruktoren bereit, welche die Parameter der entsprechenden Konstruktoren von Mail besitzen und zusätzlich zwei weitere Parameter für das Gewicht und den Versicherungsstatus des Parcel. Letzterer Parameter habe den Defaultwert false.

Spezialisieren Sie die Basisklassen-Methode delivery_text() in der Klasse Parcel (unter Nutzung der Basisklassenmethode), indem dort der String "Paket: " dem von der Basisklassen-Methode ausgegebenen Text vorangestellt wird.

Überladen Sie auch für die Klasse Parcel die globale Operator-Funktion <<, ohne diese als friend zu deklarieren. Ausgabe siehe Testlauf: Zeile „--- Paket-Daten ---“ gefolgt von Zeile mit Gewicht und Versicherungsstatus gefolgt von den Ausgaben für Mail: Mail-ID, Absender, Empfänger, delivery_text.

a) Erstellen Sie das UML-Klassendiagramm für die Klasse Mail und ihre abgeleiteten Klassen Letter und Parcel.

b) Programmieren Sie in einer vollständigen Headerdatei Parcel.h den C++ Code der Klasse Parcel.

Nutzen Sie das folgende Hauptprogramm:

```
// Datei: main.cpp
```

```
#include <iostream>
using namespace std;

#include "Address.h"
#include "Mail.h"
#include "Letter.h"
#include "Parcel.h"

int main()
{
    // --- Briefe ---
    Address to("Boss, Anton", "Antonstr. 11", "23456 Hamburg");

    // Zwei Briefe ...
    Letter letter1(102030, EXPRESS),
        letter2(203040, Address{}, to, STANDARD);

    cout << "Die Briefe:" << endl << letter1 << letter2 << endl;

    cout << "Bitte Absender fuer Brief " << letter1.get_id()
         << " eingeben:" << endl;
    if (!letter1.scan_from())
        cout << "Ungueltige Eingabe!" << endl;

    cout << "und der Empfaenger:" << endl;
    if (!letter1.scan_to())
        cout << "Ungueltige Eingabe!" << endl;

    cout << "Die neuen Daten des Briefs: \n" << letter1 << endl;

    cout << "Ein Versuch, den Brief auszuliefern:" << endl;
    if (letter1.deliver())
        cout << "Brief mit der ID " << letter1.get_id()
             << " wurde ausgeliefert!\n" << endl;
    else
        cout << "Kein gueltiger Empfaenger!" << endl;

    // --- Pakete ---
    Parcel parcel1(500001, 2.5, true),
        parcel2(500002, Address("Bauer, Hans", "Feldweg 2",
                "88888 Einoed"), to, 10.7);

    cout << "Die Pakete:" << endl << parcel1 << parcel2 << endl;

    cout << "Bitte Empfaenger fuer Paket " << parcel1.get_id()
         << " eingeben:" << endl;
    if (!parcel1.scan_to())
        cout << "Ungueltige Eingabe!" << endl;
    if (parcel2.deliver())
        cout << "\nPaket mit der ID " << parcel2.get_id()
```

```
            << " und dem Gewicht " << parcel2.get_weight()
            << " wurde ausgeliefert!" << endl;

    system("PAUSE");
    return 0;
}
```

Testlauf (Benutzereingaben sind unterstrichen):

```
Die Briefe:
--- Brief-Daten ---
Kategorie: Express
Mail-ID: 102030
Kein gueltiger Absender!
Kein gueltiger Empfaenger!
Noch nicht ausgeliefert!
--- Brief-Daten ---
Kategorie: Standard
Mail-ID: 203040
Kein gueltiger Absender!
Empfaenger:
Boss, Anton
Antonstr. 11
23456 Hamburg
Deutschland
Noch nicht ausgeliefert!

Bitte Absender fuer Brief 102030 eingeben:
Nachname, Vorname: Mustermann, Max
Strasse mit Hausnr.: Turmstr. 12
Postleitzahl Stadt: 12345 Aachen
Land: (leere Eingabe, d.h. ENTER gedrückt)
und der Empfaenger:
Nachname, Vorname: Mustermann, Maria
Strasse mit Hausnr.: Badstr. 34
Postleitzahl Stadt: 67890 Aachen
Land: (leere Eingabe, d.h. ENTER gedrückt)
Die neuen Daten des Briefs:
--- Brief-Daten ---
Kategorie: Express
Mail-ID: 102030
Absender:
Mustermann, Max
Turmstr. 12
12345 Aachen
Deutschland
Empfaenger:
Mustermann, Maria
Badstr. 34
```

```
67890 Aachen
Deutschland
Noch nicht ausgeliefert!

Ein Versuch, den Brief auszuliefern:
Brief mit der ID 102030 wurde ausgeliefert!

Die Pakete:
--- Paket-Daten ---
Gewicht: 2.5 / Versichert
Mail-ID: 500001
Kein gueltiger Absender!
Kein gueltiger Empfaenger!
Noch nicht ausgeliefert!
--- Paket-Daten ---
Gewicht: 10.7 / Nicht versichert
Mail-ID: 500002
Absender:
Bauer, Hans
Feldweg 2
88888 Einoed
Deutschland
Empfaenger:
Boss, Anton
Antonstr. 11
23456 Hamburg
Deutschland
Noch nicht ausgeliefert!

Bitte Empfaenger fuer Paket 500001 eingeben:
Nachname, Vorname: Mustermann, Max
Strasse mit Hausnr.: Turmstr. 12
Postleitzahl Stadt: 12345 Aachen
Land: (leere Eingabe, d.h. ENTER gedrückt)

Paket mit der ID 500002 und dem Gewicht 10.7 wurde ausgeliefert!
Drücken Sie eine beliebige Taste . . .
```

Aufgaben-Tutorium am Freitag 21.1.2022: Letzter Vorlesungstag, keine spezifische Aufgabe, allgemeiner GIP Support im Hinblick auf die GIP Prüfungen