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-Aufgaben in der Woche 15.-19.1.2024.
Abgabe der INF-13-Aufgaben in der Woche 22.-26.1.2024.
Alternativ können Sie auch alle Aufgaben INF-12 und INF-13 schon in der Woche 15.-19.1.2024 komplett abgeben. (Nur) wenn Sie dies erfolgreich tun und ihr Praktikum damit komplett erfolgreich ist, können Sie sich von der /dem für ihre Gruppe verantwortlichen Mitarbeiterin / Mitarbeiter die Erlaubnis holen, an ihrem letzten Praktikumstermin in der Woche 22.-26.1. nicht mehr erscheinen zu müssen.
Eine frühzeitige Abgabe von INF-12 oder INF-13 Aufgaben in der Woche der INF-11 Abgaben (8.-12.1.2024) ist nicht gestattet!

(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` …

Die Aufgaben können alle innerhalb des gleichen VS Code Projekts erarbeitet werden, da sie das Projekt jeweils um neue Methoden erweitern.

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

Erstellen Sie ein neues Projekt in VS Code. Laden Sie die Headerdatei `gip_mini_catch.h` aus Ilias in dieses Projekt herunter.

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

```
// Datei: main.cpp

#include <iostream>

#define CATCH_CONFIG_RUNNER
#include "gip_mini_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 { … };`

Alle Anforderungen dieser Teilaufgabe hier werden Sie wahrscheinlich noch komplett in der Headerdatei umsetzen und somit die `CharListenKnoten.cpp` Datei noch gar nicht benötigen, aber in den folgenden Teilaufgaben werden Sie dann auch die `.cpp` Datei mit Inhalten füllen…

- 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)*:

```
Alle Tests erfolgreich (6 REQUIREs 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)*:

```
Alle Tests erfolgreich (9 REQUIREs 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)*:

```
Alle Tests erfolgreich (11 REQUIREs 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)*:

```
Alle Tests erfolgreich (24 REQUIREs 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)*:

```
Alle Tests erfolgreich (29 REQUIREs 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)*:

```
Alle Tests erfolgreich (41 REQUIREs 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)*:

```
Alle Tests erfolgreich (44 REQUIREs 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)*:

```
Alle Tests erfolgreich (46 REQUIREs 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)*:

```
Alle Tests erfolgreich (53 REQUIREs 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)*:

```
Alle Tests erfolgreich (81 REQUIREs 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)*:

```
Alle Tests erfolgreich (83 REQUIREs 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)*:

```
Alle Tests erfolgreich (92 REQUIREs 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)*:

```
Alle Tests erfolgreich (94 REQUIREs 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)*:

```
Alle Tests erfolgreich (107 REQUIREs in 19 Test Cases)
Drücken Sie eine beliebige Taste . . .
```