

Praktikumstermin Nr. 08, INF: Dynamische Datenstruktur *Doppelt verkettete Liste*

Abgabe im GIP-INF Praktikum der Woche 28.11.-2.12.2022.

Vorbereitungen zur Aufgabe

Legen Sie ein Visual Studio Projekt für den vorgegebenen Programmcode an. Der vorgegebene Code in diesem Projekt realisiert die dynamische Datenstruktur *einfach verkettete Liste*, die auch in der Vorlesung vorgestellt wurde. Der vorgegebene Programmcode wird Ihnen auch in Ilias zur Verfügung gestellt.

(In der Aufgabe sollen Sie den Code dann so modifizieren bzw. erweitern, dass die resultierende Datenstruktur eine doppelt verkettete Liste bildet...)

Legen Sie eine Headerdatei `liste.h` an mit folgendem Inhalt.

```
// Datei: liste.h

#pragma once

#include <string>

struct TListenKnoten
{
    int data;
    TListenKnoten *next;
};

void hinten_anfuegen(TListenKnoten *&anker, const int wert);

std::string liste_als_string(TListenKnoten * anker);
```

Legen Sie ferner eine Datei `liste.cpp` an mit folgendem Inhalt.

```
// Datei: liste.cpp

#include <string>
#include "liste.h"

void hinten_anfuegen(TListenKnoten *&anker, const int wert)
{
    TListenKnoten *neuer_eintrag = new TListenKnoten;
    neuer_eintrag->data = wert;
    neuer_eintrag->next = nullptr;

    if (anker == nullptr)
        anker = neuer_eintrag;
    else
    {
        TListenKnoten *ptr = anker;
        while (ptr->next != nullptr)
            ptr = ptr->next;
        ptr->next = neuer_eintrag;
    }
}

std::string liste_als_string(TListenKnoten * anker)
{
    std::string resultat = "";

    if (anker == nullptr)
        return "Leere Liste.";
    else
    {
        resultat += "[ ";
        TListenKnoten *ptr = anker;
        do
        {
            resultat += std::to_string(ptr->data);

            if (ptr->next != nullptr) resultat += " , ";
            else resultat += " ";

            ptr = ptr->next;
        } while (ptr != nullptr);
        resultat += " ]";
    }

    return resultat;
}
```

Legen Sie außerdem eine Datei `liste_main.cpp` an mit folgendem Inhalt.

```
// Datei: liste_main.cpp

#include <iostream>
#include <string>

#define CATCH_CONFIG_RUNNER

#include "gip_mini_catch.h"

#include "liste.h"

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

    const size_t laenge = 10;
    TListenKnoten *anker = nullptr;

    std::cout << liste_als_string(anker) << std::endl;

    for (size_t i = 0; i < laenge; i++) {
        hinten_anfuegen(anker, i*i);
    }

    std::cout << liste_als_string(anker) << std::endl;

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

Erweitern Sie nun (als letzter Schritt der Vorbereitungen) ihr Visual Studio Code Projekt um eine Headerdatei `gip_mini_catch.h`. Übertragen Sie per Copy-Paste die Inhalte der `gip_mini_catch.h` Datei in Ilias in diese Datei. Es handelt sich um eine spezifisch für GIP erstellte "Miniatur-Variante" des in der Vorlesung vorgestellten Catch Unit Test Frameworks (Quelle: <https://github.com/catchorg/Catch2/tree/v2.x>). Diese "Miniatur-Variante" vermeidet die Probleme und langen Compilierzeiten, die unter Visual Studio Code und g++ auf Windows mit der "richtigen" `catch.h` Datei entstehen ...

Dieser Code ergibt insgesamt den folgenden Testlauf (...den Sie natürlich nicht vorzeigen müssen, da aller Programmcode ja vorgegeben war).

(Nicht vorzuzeigender) Testlauf (keine Benutzereingaben):

```
Keine Tests durchgeführt
```

```
Leere Liste.
```

```
[ 0 , 1 , 4 , 9 , 16 , 25 , 36 , 49 , 64 , 81 ]
```

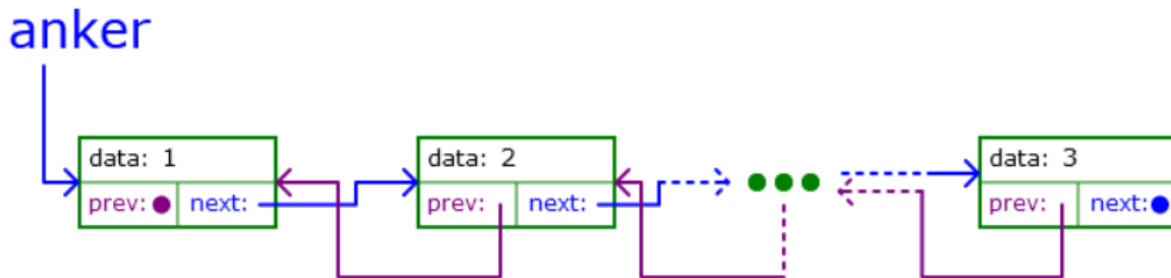
```
Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Aufgaben INF-08:

Dynamische Datenstruktur *Doppelt verkettete Liste* (Pointer, Speicherallokation auf dem Heap, Dyn. Datenstrukturen)

Erweitern Sie in den folgenden Teilaufgaben den vorgegebenen Code (sowohl die Datenstruktur als auch die Funktionen) für die dynamische Datenstruktur *einfach verkettete Liste* so, dass die resultierende Datenstruktur eine *doppelt verkettete Liste* bildet.

Bei der *doppelt verketteten Liste* zeigt jedes Listenelement sowohl auf seinen Nachfolger (Pointer `next`) als auch auf seinen Vorgänger (Pointer `prev`, von „previous“). Der erste Listenknoten hat den Nullpointer als Wert von `prev`.



Als Verankerung der Datenstruktur soll weiterhin der Pointer `anchor` auf den ersten Listenknoten verwendet werden, auch wenn dadurch die doppelte Verkettung nicht viel Nutzen bringt.

Bei allen folgenden Teilaufgaben müssen Sie letztendlich nur das Ergebnis der letzten Teilaufgabe vorzeigen, da dort die Ergebnisse aller vorherigen Teilaufgaben enthalten sind ...

(Pflicht-) Teil-Aufgabe INF-08.01: Dynamische Datenstruktur "Doppelt verkettete Liste": Funktion `hinten_anfuegen()` modifizieren

Ändern Sie in der Datei `liste.h` die Definition der Datenstruktur `TListenKnoten` so, dass die Datenstruktur auch einen Pointer `prev` auf einen Vorgängerknoten vom Typ `TListenKnoten` speichern kann.

Erweitern Sie ihr Visual Studio Projekt dann um die Datei `test_hinten_anfuegen.cpp`, die in Ilias zu finden ist (leere Datei `test_hinten_anfuegen.cpp` anlegen, Inhalt per copy-paste kopieren).

Ändern Sie die Funktion `hinten_anfuegen()` so, dass auch der `prev` Pointer der Listenknoten jeweils korrekt gesetzt wird, wenn ein neuer Knoten hinten an die (jetzt doppelt verkettete) Liste angehängt wird.

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

Alle Tests erfolgreich (24 REQUIRES in 3 Test Cases)

Leere Liste.

[0 , 1 , 4 , 9 , 16 , 25 , 36 , 49 , 64 , 81]

Drücken Sie eine beliebige Taste . . .

(Pflicht-) Teil-Aufgabe INF-08.02: Dynamische Datenstruktur "Doppelt verkettete Liste": Funktion `rueckwaerts_ausgeben()`

Erweitern Sie ihr Visual Studio Projekt dann um die Datei `test_liste_als_string_rueckwaerts.cpp`, die in Ilias zu finden ist (leere Datei `test_liste_als_string_rueckwaerts.cpp` anlegen, Inhalt per copy-paste kopieren).

Fügen Sie der Datei `liste.cpp` eine neue Funktion ...

```
std::string liste_als_string_rueckwaerts(TListenKnoten* anker)
```

... hinzu, welche die Liste "rückwärts gelesen" als String zurückgibt. Die Funktion soll sich ausgehend von `anker` erst bis zum Ende der Liste „durchhangeln“ und dann die `prev` Verkettung in Rückrichtung bei der Ermittlung des Ergebnis-Strings nutzen.

Fügen Sie der Headerdatei `liste.h` auch den Prototypen der Funktion hinzu.

Erweitern Sie das Hauptprogramm wie folgt um den Aufruf dieser Funktion:

```
int main()
{
    size_t laenge = 10;
    TListenKnoten *anker = nullptr;

    std::cout << liste_als_string(anker) << std::endl;
    std::cout << liste_als_string_rueckwaerts(anker) << std::endl; // neu

    for (size_t i = 0; i < laenge; i++) {
        hinten_anfuegen(anker, i*i);
    }

    std::cout << liste_als_string(anker) << std::endl;
    std::cout << liste_als_string_rueckwaerts(anker) << std::endl; // neu

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

(Nicht vorzuzeigender) Testlauf (keine Benutzereingaben):

Alle Tests erfolgreich (27 REQUIRES in 6 Test Cases)

Leere Liste.

Leere Liste.

[0 , 1 , 4 , 9 , 16 , 25 , 36 , 49 , 64 , 81]

[81 , 64 , 49 , 36 , 25 , 16 , 9 , 4 , 1 , 0]

Drücken Sie eine beliebige Taste . . .

(Pflicht-) Teil-Aufgabe INF-08.03: Dynamische Datenstruktur "Doppelt verkettete Liste": Funktion `in_liste_einfuegen()`

Erweitern Sie ihr Visual Studio Projekt dann um die Datei `test_in_liste_einfuegen.cpp`, die in Ilias zu finden ist (leere Datei `test_in_liste_einfuegen.cpp` anlegen, Inhalt per copy-paste kopieren).

Fügen Sie der Datei `liste.cpp` eine neue Funktion ...

```
void in_liste_einfuegen(TListenKnoten* &anker,
                       int wert_neu,
                       int vor_wert)
```

... hinzu, welche einen neuen Wert `wert_neu` in die Liste einfügt, und zwar vor der Stelle des ersten Vorkommens des Wertes `vor_wert`.

Sollte der Wert `vor_wert` nicht in der Liste vorkommen, so soll `wert_neu` ans Ende der Liste angehängt werden.

Die Funktion `in_liste_einfuegen()` soll auch in der Lage sein, einen Wert in eine bisher leere Liste einzufügen.

Fügen Sie der Headerdatei `liste.h` auch den Prototypen der Funktion hinzu.

Erweitern Sie das Hauptprogramm wie folgt:

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

    const size_t laenge = 10;
    TListenKnoten *anker = nullptr;

    std::cout << liste_als_string(anker) << std::endl;
    std::cout << liste_als_string_rueckwaerts(anker) << std::endl;

    for (size_t i = 0; i < laenge; i++) {
        hinten_anfuegen(anker, i*i);
    }

    std::cout << liste_als_string(anker) << std::endl;
    std::cout << liste_als_string_rueckwaerts(anker) << std::endl;

    // neu Aufgabe INF-08.03
    in_liste_einfuegen(anker, 11, 0); // neu
    std::cout << liste_als_string(anker) << std::endl; // neu
    std::cout << liste_als_string_rueckwaerts(anker) << std::endl; // neu
```



```
in_liste_einfuegen(anker, 22, 25); // neu
std::cout << liste_als_string(anker) << std::endl; // neu
std::cout << liste_als_string_rueckwaerts(anker) << std::endl; // neu

in_liste_einfuegen(anker, 33, 81); // neu
std::cout << liste_als_string(anker) << std::endl; // neu
std::cout << liste_als_string_rueckwaerts(anker) << std::endl; // neu

in_liste_einfuegen(anker, 44, 99); // neu
std::cout << liste_als_string(anker) << std::endl; // neu
std::cout << liste_als_string_rueckwaerts(anker) << std::endl; // neu

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

(Nicht vorzuzeigender) Testlauf (keine Benutzereingaben):

Alle Tests erfolgreich (185 REQUIREs in 18 Test Cases)

```
Leere Liste.
Leere Liste.
[ 0 , 1 , 4 , 9 , 16 , 25 , 36 , 49 , 64 , 81 ]
[ 81 , 64 , 49 , 36 , 25 , 16 , 9 , 4 , 1 , 0 ]
[ 11 , 0 , 1 , 4 , 9 , 16 , 25 , 36 , 49 , 64 , 81 ]
[ 81 , 64 , 49 , 36 , 25 , 16 , 9 , 4 , 1 , 0 , 11 ]
[ 11 , 0 , 1 , 4 , 9 , 16 , 22 , 25 , 36 , 49 , 64 , 81 ]
[ 81 , 64 , 49 , 36 , 25 , 22 , 16 , 9 , 4 , 1 , 0 , 11 ]
[ 11 , 0 , 1 , 4 , 9 , 16 , 22 , 25 , 36 , 49 , 64 , 33 , 81 ]
[ 81 , 33 , 64 , 49 , 36 , 25 , 22 , 16 , 9 , 4 , 1 , 0 , 11 ]
[ 11 , 0 , 1 , 4 , 9 , 16 , 22 , 25 , 36 , 49 , 64 , 33 , 81 , 44 ]
[ 44 , 81 , 33 , 64 , 49 , 36 , 25 , 22 , 16 , 9 , 4 , 1 , 0 , 11 ]
Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Teil-Aufgabe INF-08.04: Dynamische Datenstruktur "Doppelt verkettete Liste": Funktion `aus_liste_loeschen()`

Erweitern Sie ihr Visual Studio Projekt dann um die Datei `test_aus_liste_loeschen.cpp`, die in Ilias zu finden ist (leere Datei `test_aus_liste_loeschen.cpp` anlegen, Inhalt per copy-paste kopieren).

Fügen Sie der Datei `liste.cpp` eine neue Funktion ...

```
void aus_liste_loeschen(TListenKnoten* &anker, int wert)
```

... hinzu, welche in der Liste den *ersten* Knoten mit Wert `wert` löscht und den `anker`, falls die Liste dadurch leer wird, auf den `nullptr` Wert zurücksetzt. Sollten mehrere Knoten mit dem Wert `wert` in der Liste vorkommen, so werde *nur der Erste* dieser Knoten gelöscht.

Die Funktion `aus_liste_loeschen()` soll auch in der Lage sein, für eine leere Liste aufgerufen zu werden, als auch für eine Liste, in welcher der Wert `wert` gar nicht vorkommt. In diesen Fällen soll letztendlich nichts passieren, da kein Knoten zu löschen ist.

Beachten Sie auch, was passieren muss, wenn der Wert `wert` im ersten Knoten der Liste vorkommt.

Fügen Sie der Headerdatei `liste.h` auch den Prototypen der Funktion hinzu.

Verwenden Sie jetzt folgendes weiter modifizierte Hauptprogramm:

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

    const size_t laenge = 10;
    TListenKnoten *anker = nullptr;

    std::cout << liste_als_string(anker) << std::endl;
    std::cout << liste_als_string_rueckwaerts(anker) << std::endl;

    for (size_t i = 0; i < laenge; i++) {
        hinten_anfuegen(anker, i*i);
    }

    std::cout << liste_als_string(anker) << std::endl;
    std::cout << liste_als_string_rueckwaerts(anker) << std::endl;

    in_liste_einfuegen(anker, 11, 0);
```

```
std::cout << liste_als_string(anker) << std::endl;
std::cout << liste_als_string_rueckwaerts(anker) << std::endl;

in_liste_einfuegen(anker, 22, 25);
std::cout << liste_als_string(anker) << std::endl;
std::cout << liste_als_string_rueckwaerts(anker) << std::endl;

in_liste_einfuegen(anker, 33, 81);
std::cout << liste_als_string(anker) << std::endl;
std::cout << liste_als_string_rueckwaerts(anker) << std::endl;

in_liste_einfuegen(anker, 44, 99);
std::cout << liste_als_string(anker) << std::endl;
std::cout << liste_als_string_rueckwaerts(anker) << std::endl;

// Neu Aufgabe INF-08.04
aus_liste_loeschen(anker, 11); // neu
std::cout << liste_als_string(anker) << std::endl; // neu
std::cout << liste_als_string_rueckwaerts(anker) << std::endl; // neu

aus_liste_loeschen(anker, 22); // neu
std::cout << liste_als_string(anker) << std::endl; // neu
std::cout << liste_als_string_rueckwaerts(anker) << std::endl; // neu

aus_liste_loeschen(anker, 33); // neu
std::cout << liste_als_string(anker) << std::endl; // neu
std::cout << liste_als_string_rueckwaerts(anker) << std::endl; // neu

aus_liste_loeschen(anker, 44); // neu
std::cout << liste_als_string(anker) << std::endl; // neu
std::cout << liste_als_string_rueckwaerts(anker) << std::endl; // neu

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

Vorzuzeigender Testlauf (keine Benutzereingaben):

Alle Tests erfolgreich (272 REQUIREs in 30 Test Cases)

Leere Liste.

Leere Liste.

```
[ 0 , 1 , 4 , 9 , 16 , 25 , 36 , 49 , 64 , 81 ]
[ 81 , 64 , 49 , 36 , 25 , 16 , 9 , 4 , 1 , 0 ]
[ 11 , 0 , 1 , 4 , 9 , 16 , 25 , 36 , 49 , 64 , 81 ]
[ 81 , 64 , 49 , 36 , 25 , 16 , 9 , 4 , 1 , 0 , 11 ]
[ 11 , 0 , 1 , 4 , 9 , 16 , 22 , 25 , 36 , 49 , 64 , 81 ]
[ 81 , 64 , 49 , 36 , 25 , 22 , 16 , 9 , 4 , 1 , 0 , 11 ]
[ 11 , 0 , 1 , 4 , 9 , 16 , 22 , 25 , 36 , 49 , 64 , 33 , 81 ]
[ 81 , 33 , 64 , 49 , 36 , 25 , 22 , 16 , 9 , 4 , 1 , 0 , 11 ]
[ 11 , 0 , 1 , 4 , 9 , 16 , 22 , 25 , 36 , 49 , 64 , 33 , 81 , 44 ]
[ 44 , 81 , 33 , 64 , 49 , 36 , 25 , 22 , 16 , 9 , 4 , 1 , 0 , 11 ]
[ 0 , 1 , 4 , 9 , 16 , 22 , 25 , 36 , 49 , 64 , 33 , 81 , 44 ]
[ 44 , 81 , 33 , 64 , 49 , 36 , 25 , 22 , 16 , 9 , 4 , 1 , 0 ]
[ 0 , 1 , 4 , 9 , 16 , 25 , 36 , 49 , 64 , 33 , 81 , 44 ]
[ 44 , 81 , 33 , 64 , 49 , 36 , 25 , 16 , 9 , 4 , 1 , 0 ]
[ 0 , 1 , 4 , 9 , 16 , 25 , 36 , 49 , 64 , 81 , 44 ]
[ 44 , 81 , 64 , 49 , 36 , 25 , 16 , 9 , 4 , 1 , 0 ]
[ 0 , 1 , 4 , 9 , 16 , 25 , 36 , 49 , 64 , 81 ]
[ 81 , 64 , 49 , 36 , 25 , 16 , 9 , 4 , 1 , 0 ]
Drücken Sie eine beliebige Taste . . .
```

Nur dieser letzte Testlauf muss im Praktikum vorgezeigt werden, da er alle vorherigen Operationen beinhaltet.