Hochschule Darmstadt
Prof. Dr. Benjamin Meyer

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

13. Mai 2021

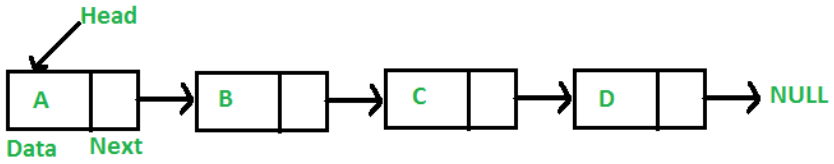
Programmieren, Algorithmen und Datenstrukturen II, SoSe 2021
Aufgabenblatt 3

3.1 Einfach verkettete Liste

Die Containerklassen aus der STL sind ein hilfreiches Werkzeug in vielen unterschiedlichen Anwendungsgebieten. Um sie effizient nutzen zu können, muss die interne Funktionsweise allerdings gut verstanden sein. Neben Stacks, Queues und Deques sind aus der Vorlesung auch einfach verkettete Listen bekannt. Ihre Aufgabe soll es nun sein, eine solche Liste als objekt-orientiertes Klassen-Template selber zu schreiben.
Sie benötigen hierfür eine kleine Klasse `Item`, welches die später zu speichernden Elemente repräsentieren soll. Wie in der Vorlesung besprochen, enthält das `Item` vor allem zwei Attribute: die eigentliche Information, sowie eine Referenz auf das folgende Element. Die gespeicherte Information soll dabei jeden beliebigen Datentyp annehmen können und nicht im Vorfeld festgelegt sein müssen.
Eine zweite Klasse stellt die eigentliche `Liste` dar. Die `Liste` speichert sich einen Zeiger auf das erste Element sowie einen Positionszeiger auf das aktuelle Element, und bietet dem Nutzer dabei die folgenden Funktionalitäten an:

- get(), liefert das aktuelle Element, auf das der Positionszeiger zeigt
- empty(), Liste leer?
- end() , Ende erreicht?
- adv(), Positionszeiger vorrücken
- reset(), Positionszeiger an den Anfang setzen
- ins(), am Positionszeiger ein neues Element hinzufügen
- del(), am Positionszeiger das Element löschen

Der Zeiger des letzten Elements der Liste (also des letzten Objektes vom Typ Item) wird als nullptr gesetzt, zeigt also auf NULL. Ist die Liste leer, zeigt bereits der Anfangszeiger auf NULL. Die Struktur können Sie sich ähnlich der folgenden Abbildung vorstellen.



Zu Testzwecken soll die Liste zudem eine Methode `print()` enthalten, die den kompletten Inhalt der Liste ausgibt. Hierfür bietet es sich an, dem `Item` bereits eine `print()`-Methode zu spendieren und diese

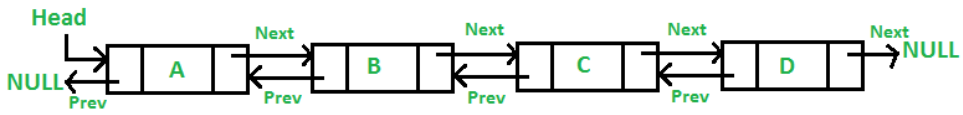
dann für jedes `Item` der `Liste` aufzurufen. Der Positionszeiger der Liste soll nach der Ausgabe allerdings weiterhin/wieder auf der "alten" Position stehen.
Implementieren Sie zudem ein minimalistisches Input-Menu, um mit der Liste arbeiten zu können (übernehmen Sie ruhig den Quellcode vom letzten Praktikum für das Abfragen von Benutzereingaben, um Zeit zu sparen).

3.2 UnitTests

Die Methoden `ins()` und `del()` der Liste sind die kompliziertesten, weil dort die Zeiger der Elemente "umgebogen" werden müssen. Erzeugen Sie für diese beiden Methoden UnitTests, um sicherzustellen, dass an den richtigen Stellen eingefügt und gelöscht wird. Betrachten Sie auch die entsprechenden Randfälle (einfügen/löschen am Anfang/Ende). Werfen Sie zudem eine entsprechende Ausnahme, wenn versucht wird, aus einer leeren Liste noch ein Element zu löschen.
Anmerkung: Für die UnitTests wird die Nutzung der QUnitTests empfohlen, da diese in der Vorlesung besprochen wurden. Sollten Sie andere UnitTests bevorzugen, müssen Sie sich eigenständig in die Benutzung einarbeiten und im Praktikum in der Lage sein, die Funktionsweise der Tests zu erklären.

3.3 Doppelt verkettete Liste

Die Standard-Liste der STL basiert zwar auf demselben Konzept der `Liste`, ist aber etwas komplizierter aufgebaut. Neben einem Zeiger auf das jeweils nächste Element gibt es in dieser sog. doppelt verketteten Liste (eng. Doubly Linked List) auch einen Zeiger auf das vorherige Element. Passen Sie Ihre `Liste`-Implementierung entsprechend an. Verbessern Sie neben der `Item`-Klasse vor allem die Methoden `ins()` und `del()`, damit dort beide Zeiger entsprechend gesetzt werden. Die folgende Abbildung zeigt die so entstehende Datenstruktur.



Die STL-Liste hat jedoch noch eine weitere Eigenschaft: um schnell auf die hinteren Elemente der Liste zugreifen zu können, wird neben dem Anfangszeiger auf das erste Element noch ein Endzeiger auf das letzte Element der Liste gespeichert (zu Beginn ist auch dieser auf nullptr gesetzt). Integrieren Sie dieses Feature in Ihre Implementierung. Passen Sie auch die geschriebenen UnitTests an die neue Situation an – zeigen weiterhin alle Zeiger auf die richtigen Elemente nach ins() oder del()?
Können Sie grob abschätzen, was dies für die Laufzeit der Einfüge- und Löschoperationen auf der Liste bedeutet?

3.4 Sortierte Liste

Ein später äußerst nützlicher Spezialfall der Linked List ist die sog. Sorted List. Bis jetzt lässt sich unsere Liste jedoch nicht sortieren, da keine logische Beziehung zwischen den gespeicherten Elementen besteht. Dies können Sie ändern, indem Sie in der `Item`-Klasse die Vergleichsoperatoren `<`, `>` und `==` überladen. Ein Beispiel hierfür finden Sie im Vorlesungsskript – allerdings gilt es eine Sache zu beachten: Die Überladung von Operatoren ist keine direkte Klassen-Funktion. Sie könnten daher die überladenen Vergleichsoperatoren außerhalb der Klasse implementieren, oder lassen sie zwar in der Klasse Item stehen, versehen sie aber zusätzlich mit dem Keyword `friend`. Dadurch werden die Methoden non-member-Methoden (also nicht mehr an die Klassen-Objekte gebunden), die jedoch trotzdem Zugriff auf alle (privaten) Klassenattribute bekommen.
Testen Sie Ihre Implementierung, können Sie nun zwei Objekte des Typs Item miteinander vergleichen? Prima, dann können Sie im letzten Schritt nun die `ins()`-Methode anpassen, sodass unabhängig von dem Positionszeiger der Liste immer an der semantisch richtigen Stelle eingefügt wird, sodass die einzelnen Elemente der Liste sortiert bleiben. Beim Löschen eines Elements ändert sich natürlich nichts.