Dirk Praetorius,
Marcus Page,
Thomas Führer

Wintersemester 2012/13
09.01.2013

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 13

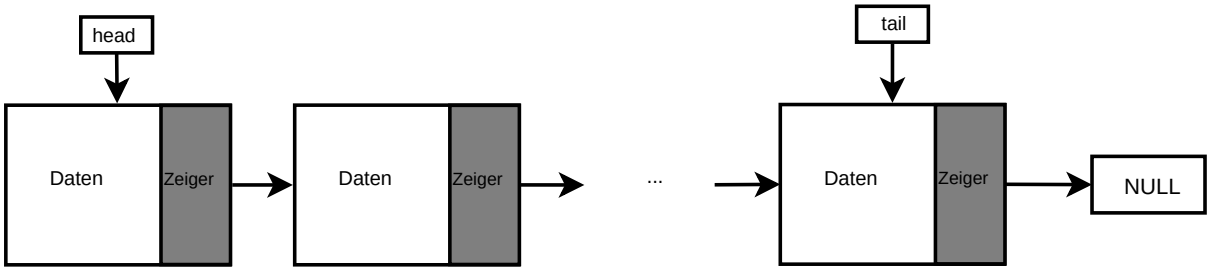
Aufgabe 13.1*. *(Vererbung)* Welche Vorteile bietet das Konzept der Vererbung? Erklären Sie den Unterschied zwischen Redefinition und Überlagerung von Methoden. Was ist beim redefinieren einer Methode zu beachten?

Aufgabe 13.2*. *(Vererbung)* Schreiben Sie eine Klasse `symmMatrix` zur Speicherung symmetrischer, quadratischer Matrizen. Die Speicherung soll selbstverständlich effizient sein, d.h. doppelte Einträge (aufgrund von Symmetrie) sollen nur einmal gespeichert werden. Wieviel Speicher benötigt eine solche Matrix?
Um Arbeit zu sparen soll die Klasse natürlich von `Matrix` aus der Vorlesung abgeleitet werden. Was müssen Sie in der Basisklasse beachten, damit etwaige Funktionalität in der abgeleiteten Klasse vorhanden ist?

Aufgabe 13.3*. *(SPD-Matrizen)* In der numerischen Mathematik haben wir es sehr häufig mit sogenannten *SPD Matrizen* zu tun. Das sind symmetrische Matrizen die zusätzlich noch *positiv definit* sind. Hierfür gibt es eine ganze Reihe spezieller Algorithmen.
Schreiben Sie eine Klasse `spdMatrix` die Sie von `symmMatrix` ableiten. Müssen Sie hierbei in der Basisklasse (= `symmMatrix`) etwas spezielles beachten?
Zu jeder SPD-Matrix $A \in \mathbb{R}^{n\times n}$ gibt es eine untere Dreiecksmatrix $L \in \mathbb{R}^{n\times n}$ mit Diagonalelementen $\ell_{jj} > 0$, so dass $A = LL^T$ gilt (d.h. A besitzt eine spezielle LU-Zerlegung). Leiten Sie aus der Formel für das Matrixprodukt $A = LL^T$ eine Formel für die Einträge von L her und schreiben Sie eine Methode `chol` von `spdMatrix` die L zurückliefert. Speichern Sie den Rückgabewert mit einem sinvollen Datentyp aus der Vorlesung.

Aufgabe 13.4*. *(Suchen in sortierten Arrays)* Man arbeitet gerne mit sortierten Arrays, weil man Einträge darin sehr effizient finden kann. Die simple *lineare* Suche, bei der man das Array einfach von vorne bis hinten durchgeht ist hierfür allerdings nicht geeignet.
Wenn man einen Namen innerhalb eines Telefonbuches sucht, dann würde man ja auch nicht das ganze Telefonbuch von vorne durchlesen bis man beim entsprechenden Namen angekommen ist. Vielmehr würde man eine zufällige Seite aufschlagen und nachschauen ob der gesuchte Name weiter vorne oder weiter hinten zu finden ist. Dieses Suchverfahren nennt man binäres Suchen.
Überlegen Sie sich, wie man die oben skizzierte binäre Suche formalisieren kann und implementieren Sie eine Funktion `binarySearch`, die ein sortiertes Array effizient mittels binärer Suche nach einem entsprechenden Eintrag durchsucht. Beachten Sie hierbei explizit den Fall, dass die Suche auch leer ausgehen kann, d.h. dass sich der gesuchte Eintrag garnicht im Array befindet. Welche Laufzeit ist für diesen Algorithmus zu erwarten? Verwenden Sie die $\mathcal{O}$-Notation und begründen Sie Ihre Antwort.

Aufgabe 13.5. *(Einfach verkettete Listen)*



In dieser Aufgabe lernen wir einen neuen Datentypen kennen, die einfach verkettete Liste (siehe Abbildung). Diese Liste besteht aus mehreren Elementen (= *Knoten*). Jedes Element besteht dabei aus zwei Teilen, einem Datenanteil und einem Zeiger auf das nächste Listenelement. Die Liste wird durch den Kopf (= *head*) beschrieben, welcher auf das erste Element in der Liste zeigt. Das letzte Element (= *tail*) zeigt auf `NULL`. Laden Sie das Paket `liste.zip` von der EPROG Homepage herunter. Im Quellcode sind die zwei Klassen `ListenElement` und `Liste` definiert. Die Klasse `ListenElement` dient zur Beschreibung eines Knotens einer einfachen Liste. Die Klasse `Liste` enthält die Membervariablen `head, tail` und zusätzlich `size`. Weiters sind bereits Methoden implementiert um Daten (in diesem Fall handelt es sich um einen einfachen Integerwert) am Anfang bzw. am Ende einzufügen, zu löschen oder auszulesen. Bauen Sie an entsprechenden Stellen Sicherheitsabfragen ein. Erstellen Sie ein Hauptprogramm und testen Sie den Code. Implementieren Sie eine Methode `insertAfterPos(int pos, Daten d)` welche ein neues Listenelement nach einer gewissen Position `pos` in die Liste einfügt. Ein Vorteil von einfachen Listen gegenüber Arrays besteht darin, dass man relativ leicht Elemente einfügen kann, ohne einen Teil der anderen Elemente umkopieren zu müssen (im Prinzip versetzt man nur Pointer). Weiters muss der Speicherbereich der einzelnen Knoten nicht zusammenhängend im Speicher sein, wie das bei Arrays der Fall ist. Welche Nachteile sehen Sie bei der Verwendung von Listen?

Aufgabe 13.6. *(Einfach verkettete Listen, Aufwand)* Der Aufwand um ein Element am Anfang einer Liste einzufügen oder zu löschen ist konstant, d.h. $\mathcal{O}(1)$. Wie groß ist der Aufwand um ein Element am Ende der Liste einzufügen und zu löschen? Wie groß ist der Aufwand um ein Element an einer beliebigen Position einzufügen? Überlegen Sie auch wie groß der Aufwand ist um am Anfang eines Arrays einen Wert einzufügen bzw. zu löschen. Schreiben Sie ein Testprogramm indem Sie ihre Ergebnisse testen (Zeitmessung und Tabelle erstellen).

Aufgabe 13.7. *(Doppelt verkettete Listen)* Eine doppelt verkettete Liste besteht wie die einfach verkettete Liste aus Knoten. Zusätzlich zu dem Zeiger auf den Nachfolger besitzt ein Element der Liste auch einen Zeiger auf seinen Vorgänger in der Liste (der Vorgänger des Kopfes ist dabei der `NULL`-Zeiger). Erweitern Sie die Programmcodes von der EPROG-Homepage, so dass die Klasse `Liste` eine doppelt verkettet Liste beschreibt.
Welche Vorteile sehen Sie in der Verwendung doppelt verketteter Listen? Welche Nachteile gibt es?
Hinweis: Fügen Sie zunächst zur Klasse `ListenElement` einen zusätzlichen Zeiger hinzu, welcher auf das Vorgängerelement zeigen soll. Ändern Sie danach in den Klassenmethoden von `Liste` an den entsprechenden Stellen den Quellcode.

Aufgabe 13.8. *(Sortieren einfach verketteter Listen)* Implementieren Sie eine Methode `sort` von `Liste`, die eine gegebene einfach verkettete Liste sortiert. Verwenden Sie hierzu den *Bubblesort* Algorithmus aus der Vorlesung.
Ein großer Nachteil einfach verketteter Listen besteht darin, dass es auch in sortierten Listen lange dauern kann, einen Eintrag zu suchen. Woran liegt das? Implementieren Sie eine Mehtode `searchEntry`, die einen gegebenen Eintrag sucht und vergleichen Sie die Laufzeiten mit Ihren Ergebnissen aus Aufgabe 13.4.