Programmierübungen

Wintersemester 2006/2007

8. Übungsblatt
15. Dezember 2006

Abgabe bis Samstag, 6. Januar 23:59 Uhr.

Die Abgabe Ihrer Bearbeitung können Sie im eClaus-System durchführen. Erarbeiten Sie Lösungsideen zu den Aufgaben möglichst in Kleingruppen. Es wird jedoch von Ihnen erwartet, dass jeder Teilnehmer eine eigene Lösung abgibt. Sollten kopierte Quelltexte abgegeben werden, so werden grundsätzlich alle Kopien mit 0 Punkten bewertet. In den Vortragsfolien der Programmierübungen oder im Skript zur Einführung in die Informatik abgedruckte Quelltexte können verwendet werden, müssen aber der Programmierrichtlinie entsprechend formatiert und kommentiert werden.

Beachten Sie die Programmierrichtlinie und kommentieren Sie Ihren Quelltext. Dokumentieren Sie unbedingt Ihre Lösungsidee in den Quelltext-Kommentaren.

`http://www.iste.uni-stuttgart.de/ps/Lehre/WS0607/inf-prokurs`

Am Montag 18.12.2006, 8:00 Uhr in V38.01 wird das Aufgabenblatt kurz vorgestellt und Sie haben Gelegenheit Fragen zu den Aufgaben zu stellen.

Aufgabe 8.1: Einfach verkettete Listen (8 Punkte)

Im Skript zur Vorlesung „Einführung in die Informatik 1“ wird in Abschnitt 3.5.2.0 ein Rahmenprogramm für die Behandlung von einfach verketteten Listen angegeben. Setzen Sie diesen Rahmen (das Programm Anwendung) in ein eigenes Programm um. Verwenden Sie als Typ Item den Typ Integer. Nennen Sie Ihr Programm Integer_Liste. Sie müssen in Ihr Programm nur die Operationen einfügen, die im folgenden tatsächlich verwendet werden. Achten Sie darauf, dass der Speicher für Listenzellen, die nicht mehr über Zeiger erreichbar sind, freigegeben wird.

Erweitern Sie das Programm so, dass als Kommandozeilen-Argument der Name einer Textdatei übergeben werden kann. Die Textdatei enthält nur ganze Zahlen, getrennt durch Leerzeichen. Das Programm liest die Textdatei und speichert alle Werte in einer Liste.

Implementieren Sie folgende weitere Operationen:

- Umkehren der Reihenfolge der Listeneinträge,
- Sortieren der Liste.
- Zerstören der Liste. Hier muss besonders darauf geachtet werden, dass der für die Listenzellen allokierte Speicher wieder freigegeben wird.

Diese Operationen sollen ausschließlich durch Zeiger-Zuweisungen implementiert sein. Der Integer-Inhalt einer Listenzelle darf nach dem Einlesen der Liste nicht mehr verändert werden.

Lassen Sie das Programm dann verschiedene Ausgaben erzeugen:

1. Ausgabe der Original-Liste,

2. Ausgabe der Liste in umgekehrter Reihenfolge,

3. Ausgabe der sortierten Liste.

Beispiel (`zahlen.txt`):

```
3 4 1 6 89 -45
4 9 23 12
```

Testlauf des Programms:

```
$ integer_liste zahlen.txt
Originalliste:
3, 4, 1, 6, 89, -45, 4, 9, 23, 12
umgekehrte Liste:
12, 23, 9, 4, -45, 89, 6, 1, 4, 3
sortierte Liste:
-45, 1, 3, 4, 4, 6, 9, 12, 23, 89
```

Hinweise:

- Es reicht nicht aus, dass Ihr Programm die korrekte Ausgabe erzeugt. Die Listenoperationen müssen implementiert werden und im Quelltext erkennbar sein. Es ist nicht zulässig, die Operationen auf einem Array durchzuführen und daraus wieder eine neue Liste zu erzeugen.

- Speicher wird mit einer Instanz von Ada.Unchecked_Deallocation freigegeben. Ein Beispiel finden Sie in der Datei `person_lists.adb` aus den nächsten Aufgabe.

- Zum Sortieren der Liste können Sie folgenden Algorithmus verwenden (Selection Sort): Es soll die Quell-Liste S sortiert werden. Dazu wird eine anfangs leere Zielliste T verwendet. Der Algorithmus führt solange Schritte durch, bis S die leere Liste geworden ist. In jedem Schritt wird aus S das jeweils größte Element entfernt und an den Anfang der Liste T hinzugefügt.

Aufgabe 8.2: Doppelt verkettete Listen (10 Punkte)

Listen-Datentypen werden üblicherweise als abstrakte Datentypen gekapselt. Das hat den offensichtlichen Vorteil, dass man den Listentyp in vielen Programmeinheiten verwenden kann und trotzdem die Listen-Operationen nur einmal implementiert werden müssen. Sollte eine Änderung an der Datenstruktur notwendig werden, so muss diese nur an einer Stelle im Programm umgesetzt werden.

Auf der Webseite zu den Programmierübungen erhalten Sie die Spezifikation des Pakets Person_Lists. Als Element-Typ wurde der von Aufgabe 6.2 bereits bekannte Typ Person verwendet. Der Typ Cell_Ref stellt einen Verweis auf eine Zelle der Liste dar und dient als Listenanker. Der Typ List_Cursor modelliert eine Referenz auf ein bestimmtes Element der Liste.

Von der Webseite können Sie das Programm LPhonebook herunter laden. Dies ist eine Neuimplementierung des Programms aus der Aufgabe 6.2, worin nun eine Liste verwendet wird und somit die Begrenzung auf 100 Telefonbucheinträge wegfällt.

Implementieren Sie das Paket Person_Lists. Beachten Sie, dass die Liste zu jedem Zeitpunkt nach Name alphabetisch sortiert sein soll. Die Implementierung einiger Unterprogramme ist bereits vorgegeben. Diese können Sie verwenden oder wahlweise durch Ihre eigene ersetzen.

Es wird erwartet, dass nicht mehr benötigter Speicher (d. h. nicht erreichbare Listenzellen) freigegeben wird. Eine Instanz von Ada.Unchecked_Deallocation ist im Body des Pakets bereits vorgesehen.

Hinweise:

- In einer üblichen Listenimplementierung ist der Typ List_Cursor ein Zeiger auf eine Zelle. Um eine logische Unterscheidung zwischen Liste und Cursor zu erreichen, wurde ein separater Typ deklariert. In einem Hauptprogramm, das das Listenpaket verwendet, ist also keine Zuweisung zwischen Cell_Ref und List_Cursor erlaubt, obwohl diese technisch möglich wäre.
- Achten Sie auf die „`-- TODO:`“ Kommentare. Diese zeigen an, welche Unterprogramme noch von Ihnen geschrieben werden müssen.

Aufgabe 8.3: Fehlersuche (2 Punkte)

Das Programm Crash (das Sie auf der Webseite erhalten können) verwendet das Listenpaket aus Aufgabe 8.2. Das Programm enthält jedoch einen gravierenden Programmierfehler, der speziell im Zusammenhang mit Zeigern steht. Finden Sie den Fehler, dokumentieren Sie die Fehlerursache und korrigieren Sie das Programm. Beachten Sie, dass die Quelltext-Kommentare die Absicht des Programmierers und nicht die tatsächliche Semantik des Programms widergeben.

Zur Korrektur können Sie dem Paket Person_Lists (auch in der `person_lists.ads`) weitere Operationen hinzufügen.

Hinweise:

- Zeichnungen der tatsächlichen Zeigerziele können Ihnen helfen den tatsächlichen Programmablauf zu verstehen.
- Das konkrete Fehlerverhalten des Programms ist abhängig davon, wie Sie das Paket Person_Lists implementiert haben sowie von Version des Compilers und Betriebsystem. Beobachten Sie nicht nur das Verhalten des ausgeführten Programms, sondern untersuchen Sie den Quelltext.

Aufgabe 8.4: Fehlervermeidung (bis zu 1 Zusatzpunkt)

Für diese Aufgabe kann ein Zusatzpunkt vergeben werden, jedoch kann die Punktzahl für die Bearbeitung des gesamten Übungsblatts 20 nicht übersteigen.

Der in Aufgabe 8.3 erkannte Fehler kann durch den Compiler bereits erkannt werden. Hierzu muss die Deklaration des Typs Person_List verändert werden. Führen Sie diese Änderung durch. Dadurch wird die Übersetzung des Originalprogramms Crash mit einer Fehlermeldung abbrechen. Das Programm LPhonebook aus Aufgabe 8.2 muss aber selbstverständlich weiterhin funktionieren.

Hinweis: die Implementierung des Pakets Person_Lists braucht nicht verändert zu werden.