Prof. Dr. A. Weber
Dr. Hassan Errami

Universität Bonn
Institut für Informatik II
12.12.2016

Wintersemester 2016/2017

Übungen zur Vorlesung
Algorithmisches Denken und imperative Programmierung (BA-INF-014)
Aufgabenblatt 7

Zu bearbeiten bis: 06.01.2017

Hinweis: Mit der Bearbeitung dieses Übungszettels haben Sie bis zum 06.01.17 Zeit. Der gesamte Zettel umfasst 40 Punkte. In der nächsten Woche (19.12 - 22.12) wird die Probeklausur in den Übungen besprochen.

Aufgabe 1 *(Listen - 2+9*2=20 Punkte)*

Sie haben in der Vorlesung die Datenstruktur *IntNode* für Listen über *Integer* Zahlen sowie Funktionen für die Listenoperationen kennengelernt.
a) Betrachten Sie folgende main-Funktion und skizzieren Sie den Zustand des Speichers an Stelle 1:

```
int main(int argc, char *argv[]) {
   IntNode *L2, *L1 = NULL;
   L1 = insertFirst( L1, 3);
   L1 = insertFirst( L1, 7);
   L1 = insertFirst( L1, 11);
   L2 = L1;
   L2 = insertFirst( L2, 5);
   L2 = insertFirst( L2, 14);
   \\ Stelle 1
   printList(L1);
   printList(L2);
   return 0;
}
```

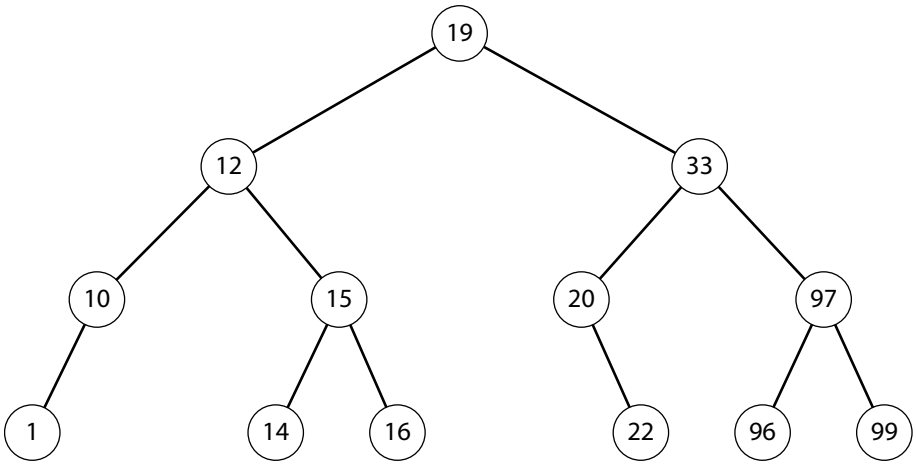
b) Implementieren Sie die entsprechende Datenstruktur *DoubleNode* für Listen über *double* Zahlen. Arbeiten Sie bei den Indizes nullbasiert.

- Implementieren Sie die Funktion *DoubleNode*insertFirst(DoubleNode*head, double c)* zum Einfügen von *double d* am Anfang einer Liste.
- Implementieren Sie die Prozedur *void printList(DoubleNode*head)*, die den Inhalt einer Liste auf der Konsole ausgibt.
- Implementieren Sie die Funktion *DoubleNode*insertLast(DoubleNode*head, double d)* zum Einfügen von *double d* am Ende einer Liste.
- Implementieren Sie die Funktion *DoubleNode*reverseDoubleListCon(DoubleNode*head)* zum konstruktiven Invertieren einer Liste.
- Implementieren Sie die Funktion *DoubleNode*reverseDoubleList(DoubleNode*head)* zum destruktiven Invertieren einer Liste.
- Implementieren Sie die Funktion *double get(DoubleNode*head, int i)*, die den Wert des i-te Elementes der Liste zurückgibt. Achten Sie auf Indexüberschreitung.
- Implementieren Sie die Funktion *DoubleNode*delete(DoubleNode*head, int i)*, die das i-te Element der Liste löscht. Achten Sie auf Indexüberschreitung.
- Implementieren Sie die Funktion *DoubleNode*insert(DoubleNode*head, double c, int i)*, die das Element c an der i-te Stelle in die Liste einfügt. Achten Sie auf Indexüberschreitung.

Bitte **testen** Sie Ihre Implementierung gründlich, auch bzgl. Randfällen! Die Funktionen sollten mit allen (gültigen) Parametern korrekt funktionieren.

Aufgabe 2 *(Baumdurchläufe - 3 Punkte)*

Gegeben sei folgender Binärbaum:



a) Geben Sie die Elemente des Baumes in der Reihenfolge an, in der sie bei einem *preorder*-Durchlauf bearbeitet werden.
b) Geben Sie die Elemente des Baumes in der Reihenfolge an, in der sie bei einem *postorder*-Durchlauf bearbeitet werden.
c) Geben Sie die Elemente des Baumes in der Reihenfolge an, in der sie bei einem *inorder*-Durchlauf bearbeitet werden.

Aufgabe 3 *(Stack - 2+5*2=12 Punkte)*

Nutzen Sie Ihre Kenntnisse über verkettete Listen, um Methoden für die Verwaltung eines Stacks zu schreiben. Ein Stack kann eine beliebige Anzahl von Elementen aufnehmen. Elemente können nur von oben auf den Stack gelegt werden (Methode *push*) und auch nur von dort wieder gelesen werden, wobei das Element dabei entfernt wird (Methode *pop*). Dies wird auch als Last-In-First-Out-Prinzip (LIFO) bezeichnet. Nehmen Sie nachfolgend an, dass Integer-Zahlen verwaltet werden sollen.

- Implementieren Sie eine passende Datenstruktur und die Methoden *push* und *pop*. Überlegen Sie sich, welchen Methodenaufrufen diese Operationen bei verketteten Listen entsprechen könnten.
- Implementieren Sie außerdem eine Methode *peek*. Diese liefert den Wert des obersten Elements ohne es zu entfernen.
- Implementieren Sie eine Methode *isempty*, die zurückgibt, ob der Stack leer ist, also keine Elemente enthält.
- Implementieren Sie eine Methode *print*, welche den Inhalt des Stacks ausgibt.
- Testen Sie Ihre Implementierung durch folgende Sequenz von Aufrufen: push(4); push(2); push(1); Ausgabe der Rückgabe von peek(); print(); Ausgabe der Rückgabe von pop(); print(); Ausgabe der Rückgabe von isempty(); pop(); pop(); Ausgabe der Rückgabe von isempty();
  Ihre Ausgaben sollten dabei etwa wie folgt aussehen: 1; 1 2 4; 1; 2 4; false; true
- Erklären Sie, welche Änderungen prinzipiell notwendig sind, um anstatt eines Stacks eine Queue (Warteschlange) zu realisieren.

Aufgabe 4 *(Bäume - 1+2+1+1=5 Punkte)*

In der Vorlesung haben Sie Baumstrukturen kennengelernt. Bearbeiten Sie folgende Aufgaben:

- Implementieren Sie eine Datenstruktur tnode, die einen Baumknoten repräsentiert und jeweils den linken und rechten Nachfolger sowie einen Integer-Wert speichert.
- Implementieren Sie Methoden für die rekursive Ausgabe in prä-, post- und inorder-Reihenfolge.
- Erzeugen Sie einen beliebigen (festgelegten) Baum in der main-Methode, welcher mindestens über 4 Knoten verfügt. Rufen Sie für diesen Ihre Methoden auf.
- Erklären Sie prinzipiell, wie Sie die Elemente des Baums in der Reihenfolge ebenenweise von links nach rechts ausgeben könnten (level-order).