Prof. Dr. A. Weber
Dr. Hassan Errami

Universität Bonn
Institut für Informatik II
30.12.2014

Wintersemester 2014/2015

Übungen zur Vorlesung
Algorithmisches Denken und imperative Programmierung (BA-INF-014)
Aufgabenblatt 8

Zu bearbeiten bis: 16.01.2015

Aufgabe 1 *(Dynamische Arrays - 12 Punkte)*

Ein dynamisches Array hat die Eigenschaft, dass man effizient am Ende des Arrays neue Elemente anfügen kann, indem man in einem statischen Array neben der Länge des statischen Arrays auch die aktuelle Größe des dynamischen Arrays speichert. Solange die aktuelle Größe kleiner als die maximale Länge ist, kann der Speicher des statischen Arrays verwendet werden.
Wenn wir an ein Array ein Element anhängen wollen, wenn die aktuelle und maximale Größe, müssen wir neuen Speicher allokieren, der die gewünschte Länge hat (oder eine größere). Die Werte aus dem alten Array müssen dann in den neuen Speicher umkopiert werden. Danach kann das neue Element hinten angefügt werden, weil wir im neuen Array bereits Speicher für dieses Element reserviert haben.
Bei der naiven Implementation des dynamischen Arrays wird jeweils Speicher der benötigten Größe allokiert; eine andere Strategie ist es, die Größe des statischen Arrays jeweils zu verdoppeln, wenn der Speicherplatz nicht ausreichend war.

a) Schreiben Sie einen Datentyp *DynArray* zur Speicherung von dynamischen *int*-Arrays. Dieser soll die aktuelle und maximale Größe des Arrays speichern und die oben skizzierte Verdopplungsstrategie realisieren. Schreiben Sie einen Datentyp *DynArrayMin*, die die Strategie realisiert, dass jeweils nur der benötigte Speicherplatz allokiert wird.
Beachten Sie, dass Sie nicht nur Speicher für eine neues Array allokieren, sondern den Speicher für das alte auch wieder freigeben, sobald dieses nicht mehr benötigt wird.
b) Implementieren Sie einen Algorithmus (*InsertSort*) zum Sortieren Einfügen in einer Folge (F). Verwenden Sie dabei in zwei Versionen die Datentypen *DynArray* und *DynArrayMin*. Vergleichen Sie die Laufzeiten Ihrer Implementierungen bei sortierten Einfügen von 1000000 Zufallszahlen.

Aufgabe 2 *(Binärer Suchbaum - 16 Punkte)*

Gegeben sei folgende Sturktur:

```
 struct knoten {
 int Datenelement;
 struct knoten *links;
 struct knoten *rechts;
 };
typedef struct knoten *Binbaum;
```

a) Schreiben Sie ein Programm unter Benutzung dieser Stuktur, das einen binären Suchbaum generiert und verarbeitet. Implementieren Sie dabei Funktionen zum:

- Einfügen eines Knotens.
- Ausgeben des Baumes in Inorder.
- Suchen nach einem Element im Baum.
- Löschen eines Knotens.

b) Legen Sie zum Testen einen Baum und legen Sie 12 Werte hinzu. Verwenden Sie dann Ihre Funktionen um nach einem Knoten zu suchen und um den Baum auszugeben.

Aufgabe 3 *(Sortieren - 12 Punkte)*

In der Vorlesung wurden die Sortieralgorithmen *Selection sort*, *Insertion sort*, *Bubble sort*, *quick sort* und *Mergesort* vorgestellt.

- Realisieren Sie arraybasierte Implementierungen für diese Algorithmen.
- Verwenden Sie diese Sortieralgorithmen (*Selection sort*, *Insertion sort*, *Bubble sort*, *quick sort* und *Mergesort*) auf 4 Arrays mit jeweils 1000, 10000, 50000 und 100000 Zufallszahlen und vergleichen Sie dabei die Laufzeiten.