Lehrstuhl für Informatik 3
Rheinisch-Westfälische Technische Hochschule
Aachen
**Univ.-Prof. Dr.-Ing. M. Nagl**

Aachen, 07. 07. 2006

**Nagl, Becker, Ranger, Wörzberger**

# Übungen zur Vorlesung „Grundgebiete der Informatik 2: Algorithmen und Programmiertechniken“

## — Blatt 12 —

Dies ist das letzte Aufgabenblatt. Es wird noch von den Übungsgruppenleitern korrigiert, die Rückgabe und Besprechung erfolgt am 18.07.2006 beim zweiten Tutoriumstermin.

## 27. Aufgabe *Generischer Puffer*: *(10 Punkte)*

In der Vorlesung und in der Fragestunde wurde ein als Klasse realisiertes generisches ADT-Modul `Buffer` vorgestellt, welcher FIFO-Semantik (“First In First Out”, auch Schlange bzw. Queue genannt) hat. Dabei wird offen gelassen, welche Typ die Einträge besitzen (z.B. `int`, `float` oder auch komplexe Typen) und wie viel Elemente gleichzeitig in dem Puffer Platz finden. Daher hat der `Buffer` einen formalen generischen Parameter `T` für den Eintragstyp und einen formalen generischen Parameter `size` für die Größe. Beide “Stellschrauben” müssen bei der Instanziierung eines Datenobjekts durch aktuelle generische Parameter festgelegt werden. Das Problem bei der bisher vorgestellten Implementierung ist, dass die Größe eines `Buffer`s bei der Instanziierung festgelegt werden muss und anschließend für das betreffende Datenobjekt nicht mehr geändert werden kann. In dieser Aufgabe soll daher ein generisches ADT-Modul implementiert werden, dass einen dynamischen Puffer `DynamicBuffer` realisiert.

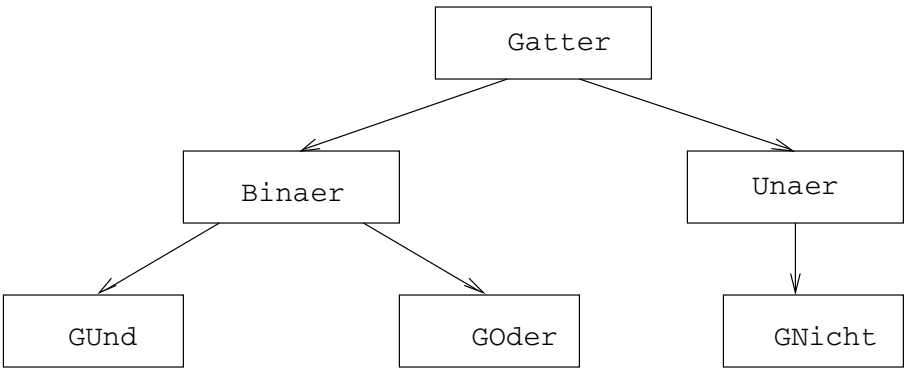
(a) Implementieren Sie das generische ADT-Modul `DynamicBuffer`. Die Schnittstelle und Rahmen für die Funktions- bzw. Prozedurdefinitionen sind dabei vorgegeben und können von der Übungsseite heruntergeladen werden (`DynamicBuffer.h` und `DynamicBuffer.cc`). Ein dynamischer Puffer kann beliebig viele Elemente aufnehmen. Daher hat der `DynamicBuffer` keinen generischen formalen Parameter `size` und bietet keine Funktion `nonFull()` an. Die interne Realisierung kann daher nicht über ein statisches Feld erfolgen. In dieser Implementierung sollen die Elemente in einer einfach verketteten Liste gespeichert werden. Hinweis: Aus technischen Gründen müssen Sie für die Verwendung des `DynamicBuffer` (beispielsweise in der `main`-Funktion) die Datei mit den Funktions- bzw. Prozedurdefinitionen inkludieren (über `#include "DynamicBuffer.cc"`) und nicht wie sonst überlich die Schnittstellen-Datei (`DynamicBuffer.h`). *(8 Punkte)*

```
#ifndef __INC_DynamicBuffer_h__
#define __INC_DynamicBuffer_h__

#include "stdio.h" // beinhaltet Definition von 'NULL'

template<class T>
class DynamicBuffer {
public:
    // Standard-Konstruktor (ohne Parameter)
    DynamicBuffer();

    // Einfuegen eines neuen Elements
    void enqueue(T x);

    // Entfernen des ältesten Elements
    void dequeue(T &x);

    // true gdw. Schlange leer ist
    bool nonEmpty();

private:
    struct ListElem; // generischer opaker Datentyp
    ListElem* head; // Listenanfang (~ ältestes Element)
    ListElem* tail; // Listenende   (~ neuestes Element)
};
#endif
```

(b) Testen Sie die Implementierung. Verwenden `DynamicBufferTest.cc` von der Web-Seite. Diese Funktion ist nicht vollständig. Es fehlen die Deklarationen für `intBuffer` und `persBuffer`. Tragen Sie diese nach und notieren/speichern Sie die Programmausgabe. *(2 Punkte)*

## 28. Aufgabe *Vererbung*: *(10 Punkte)*

Im folgenden sollen einfache logische Schaltungen simuliert werden. Dazu werden einige Klassen in C++ definiert, die diese Bauteile nachbilden. Die Basisklasse ist die Klasse `Gatter`. Weitere Schaltelemente sollen gemäß der folgenden Klassenhierarchie ebenfalls vereinbart werden:




Wir unterscheiden also Gatter mit zwei Eingängen und Gatter mit einem Eingang. Alle Gatter haben allerdings nur einen Ausgang. Um aus solchen Objekten kleine Schaltnetze aufbauen zu können, soll jede Klasse die Methoden `void setzeEingang(LogikT wert, EingangT welcher)` (setzen des Eingangswertes für Eingang `pin`) und `LogikT leseAusgang(void)` (auslesen des aktuellen Ausgabewertes) besitzen. Der Typ `LogikT` sei dabei ein Aufzählungstyp mit den Werten `LOGIK_AUS` und `LOGIK_AN`; `EingangT` ist ein diskreter Typ, der die Eingänge durchzählt.

(a) Implementieren Sie die angegebenen Klassen. Dabei sollen die Methoden `setzeEingang` und `leseAusgang` virtuell definiert werden. *(5 Punkte)*

(b) Definieren Sie eine Klasse `Halfadder`, deren Objekte sich aus den für einen Halbaddierer benötigten `Gatter`-Objekten zusammensetzen. Ein `Halfadder` soll ebenfalls Methoden zum Setzen der Eingabewerte und zum Auslesen der Ausgabewerte (allerdings hier für zwei Ausgänge) besitzen. Die Ausgabewerte sollen mit Hilfe der `Gatter`-Objekte berechnet werden. *(5 Punkte)*

## 29. Aufgabe *Operatorbaum (freiwillige Zusatzaufgabe)*: *(10 Punkte)*

Das bereits bekannte Datentypmodul `BinBaum` soll zu einem Operatorbaum erweitert werden. Nur an den Blattknoten werden Werte gespeichert, die inneren Knoten speichern *Operatoren*, mit denen der Wert des Knotens aus den Werten seiner Äste ermittelt wird.

(a) Überlegen Sie sich, welche Rolle die gespeicherten Werte an inneren Knoten haben. Sollten sie überhaupt noch zugreifbar sein? *(2 Punkte)*

(b) Implementieren Sie den Rumpf des Moduls `BinOpBaumAuswertung`. *(8 Punkte)*

Die Quelltexte der Schnittstellen sowie der Implementierungen sind wie üblich auf der Vorlesungshomepage zu haben.

---

**Abgabe: 14. 07. 2006 vor der Vorlesung. Rückgabe und Besprechung beim zweiten Tutoriumstermin..**