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“

— Lösung zu Blatt 12 —

27. Aufgabe

(a) Wichtig ist bei der Deklaration der Verbund-Komponenten, dass für den Typ der “Nutzdaten” der generische formale Parameter `T` verwendet wird. Die Deklaration der Zeiger-Komponente unterscheidet sich nicht von dem Fall einer nicht-generischen einfach verketteten Liste.

```
#ifndef __INC_DynamicBuffer_cc__
#define __INC_DynamicBuffer_cc__

#include "DynamicBuffer.h"

// Definition des generischen opaken Datentyps 'ListElem'
template<class T>
struct DynamicBuffer<T>::ListElem {
        T data;
        ListElem* next;
};

template<class T>
DynamicBuffer<T>::DynamicBuffer() {
    head = tail = NULL;
}

template<class T>
void DynamicBuffer<T>::enqueue(T x) {
    ListElem* newLE = new ListElem; // neues Element erzeugen
    newLE->data = x; // und mit Datum bestücken
    newLE->next = NULL; // ein Nächstes gibt's noch nicht

    if (nonEmpty()) { // am Ende einhängen
        tail->next = newLE;
        tail = tail->next;
    } else {
        head = newLE;
        tail = head;
    }
}

template<class T>
void DynamicBuffer<T>::dequeue(T &x) {
    if (nonEmpty()) {
```

```
            x = head->data; // Daten retten
            ListElem* toDel = head; // head-Zeiger merken
            head = head->next; // head weitersetzen
            delete toDel; // löschen
        }
    }

    template<class T>
    bool DynamicBuffer<T>::nonEmpty() {
        return (head != NULL);
    }

    #endif
```

(b) Bei den nachzutragenden Deklarationen kommt es auf die Wahl des passenden aktuellen generischen Parameters an. Im ersten Fall ist dieser `int` im zweiten Fall `Person`.

```
    ...

    int main()
    {
        DynamicBuffer<int> intBuffer;
        intBuffer.enqueue(1);

        ...

        DynamicBuffer<Person> persBuffer;
        Person pin, pout;
        pin.name = "Meier"; pin.gebJahr = 1950;
        persBuffer.enqueue(pin);

        ...

        return 0;
    }
```

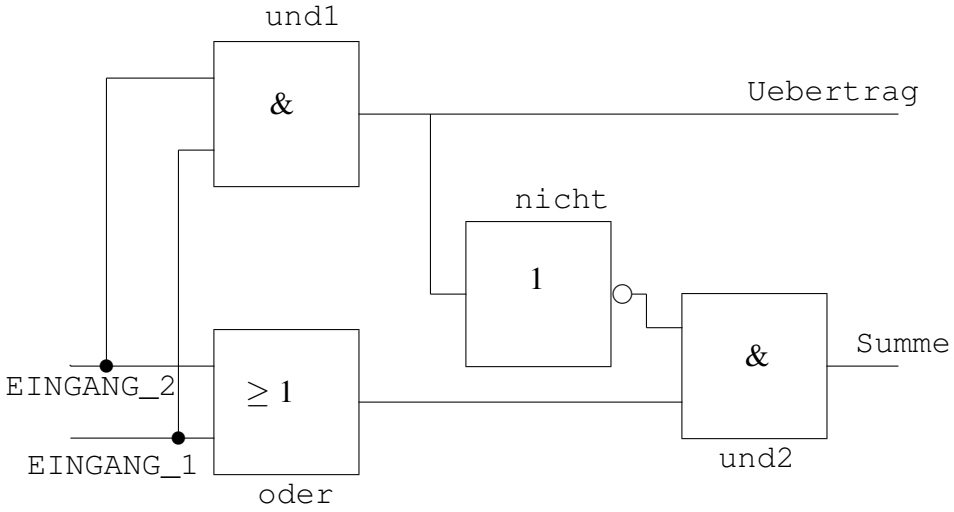
28. Aufgabe

```
#ifndef __INC_Gatter_h__
#define __INC_Gatter_h__

#include <stdlib.h>             // benutzt: NULL

enum LogikT {LOGIK_AUS = 0, LOGIK_AN = 1};
enum EingangT {EINGANG_1 = 1, EINGANG_2 = 2 };

// Ein Oberklasse, von der alle anderen Gatter-Klassen erben
class Gatter
{
public:
   /* Setzt den n-ten Eignagswertes auf den Wert wert.
      Rein virtuell, d.h. es koennen keine Objekte der
      Klasse Gatter erzeugt werden. */
   virtual void setzeEingang(LogikT wert, EingangT welcher) = NULL;

   /* Liest den Ausgangswert. */
   virtual LogikT leseAusgang(void) = NULL;
```

```
};
// Gatter mit zwei Eingaengen
class Binaer:Gatter
{
public:
    void setzeEingang(LogikT wert, EingangT welcher);
    LogikT leseAusgang(void) = NULL;
protected: LogikT ein1, ein2;
};
// Gatter mit einem Eingang
class Unaer:Gatter
{
public:
    void setzeEingang(LogikT wert, EingangT welcher=EINGANG_1);
    LogikT leseAusgang(void) = NULL;
protected: LogikT ein;
};
class GUnd  : public Binaer { public: LogikT leseAusgang(void); };
class GOder : public Binaer { public: LogikT leseAusgang(void); };
class GNicht: public Unaer  { public: LogikT leseAusgang(void); };
#endif
```

```
#include <Gatter.h>
#include <iostream.h>
void Binaer::setzeEingang(LogikT wert, EingangT welcher)
{
   if     (welcher == EINGANG_1) {ein1 = wert;}
   else if(welcher == EINGANG_2) {ein2 = wert;}
   else { cerr << "Ungueltiger Eingang." << endl; }
}
void Unaer::setzeEingang(LogikT wert, EingangT welcher)
{
   if (welcher == EINGANG_1) { ein = wert; }
   else { cerr << "Ungueltiger Eingang." << endl; }
}
LogikT GUnd::leseAusgang  (void) { return (ein1 && ein2)?LOGIK_AN:LOGIK_AUS; }
LogikT GOder::leseAusgang (void) { return (ein1 || ein2)?LOGIK_AN:LOGIK_AUS; }
LogikT GNicht::leseAusgang(void) { return (!ein        )?LOGIK_AN:LOGIK_AUS; }
```

```
#ifndef __INC_halbaddierer_h__
#define __INC_halbaddierer_h__
#include <Gatter.h>

class Halbaddierer
{
public:
    void setzeEingang(LogikT wert, EingangT welcher);
    LogikT leseSumme    (void);
    LogikT leseUebertrag(void);
private:
    GOder  oder;     GNicht nicht;     GUnd und1, und2;
};
#endif
```




```
#include <Gatter.h>
#include <Halbaddierer.h>
void Halbaddierer::setzeEingang(LogikT wert, EingangT welcher)
{
    if(welcher == EINGANG_1 || welcher == EINGANG_2) {
       und1.setzeEingang(wert, welcher);
       oder.setzeEingang(wert, welcher);
    }
}
LogikT Halbaddierer::leseSumme(void)
{
    // Interne Berechnungen
    nicht .setzeEingang(und1.leseAusgang());
    und2.setzeEingang(oder.leseAusgang() , EINGANG_1);
    und2.setzeEingang(nicht.leseAusgang(), EINGANG_2);

    return und2.leseAusgang();
}
LogikT Halbaddierer::leseUebertrag(void) { return und1.leseAusgang(); }
```

29. Aufgabe

(a) Die Werte an inneren Knoten halten Zwischenergebnisse der Berechnung. Es macht keinen Sinn, sie zu überschreiben, da sie dadurch falsche Informationen liefern könnten. Je nach Anwendung kann es jedoch interessant sein, auch die Zwischenergebnisse auszulesen (etwa in einer Lernsoftware, in der Teile einer komplizierten Formel vermittelt werden sollen). Wenn noch vermerkt wird, welche Werte seit der letzten Berechnung geändert wurden, können die Zwischenergebnisse wiederverwendet werden.

(b) Siehe Listing `LsgBinOpBaumAuswertung.cc` auf der Lehrstuhl-Website. Die Implementation durchläuft den Baum in Post-Order, um die Werte der Äste vor dem des Knotens zu berechnen.