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

Aachen, 02. 06. 2006

Nagl, Becker, Ranger, Wörzberger

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

— Lösung zu Blatt 8 —

17. Aufgabe

(a) Hauptprogramm (`hello.cc`):

```
#include "message.h" // getMessage
#include "helloio.h" // printText

int main() {
  char *message;
  message = getMessage();
  printText(message);
  return 0;
}
```

Header-Datei (`message.h`):

```
/* FM fuer Hello, World-Beispiel ================================= */
#ifndef __INC_message_h__
#define __INC_message_h__

/* Liefert einen Zeiger auf einen Text. */
char* getMessage();

#endif
/* ------------------------------------------------------------- */
```

Implementierung (`message.cc`):

```
/* body message ================================================ */

char const* getMessage() {           // Liefert einen Zeiger auf einen Text.
  return "Hello, World!";
};
/* end body message ------------------------------------------- */
```

Header Datei (`helloio.h`):

```
/* FM fuer Hello, World-Beispiel ================================= */
#ifndef __INC_HELLOIO_H__
#define __INC_HELLOIO_H__

/* Gibt einen Text aus */
void printText(char *text);

#endif
/* helloio ------------------------------------------------------ */
```

Implementierung (`helloio.cc`):

```
/* body helloio ============================================== */
#include <iostream>
using namespace std;
/* Gibt einen Text aus */
void printText(char *text) {
  cout << text << endl;
};
/* end body helloio -------------------------------------------- */
```

(b) Übersetzt werden die `*.cc`-Dateien mit `g++ -c datei.cc`, oder alle zusammen mit `g++ -c message.cc helloio.cc hello.cc`. Zum Linken dient `g++ message.o helloio.o hello.o -o hello`. Normalerweise verwendet man `Makefiles`, um die Übersetzung geänderter Sourcedateien zu steuern. Darauf kann aus Zeitgründen in der Vorlesung aber nicht weiter eingegangen werden.

18. Aufgabe

(a) Eigenschaften eines abstrakten Datentypmoduls sind z.B.:

- mehrere Instanzen (bzw. Objekte) des ADT können angelegt werden
- Funktionen und Prozeduren bekommen einen Zeiger auf eine Instanz übergeben (z.B. `push(StackTyp stack, ElementTyp element)` oder falls der ADT eine Klasse ist `stack.push(ElementTyp element))`
- Schnittstelle exportiert Instanztyp
- Schnittstelle exportiert Erzeugungsoperation mit denen Instanzen jederzeit erzeugt werden können.
- typische Merkmale: ein ModulTyp wird exportiert, Modul ist als Klasse realisiert oder Funktionen und Prozeduren bekommen alle als ersten Parameter einen Zeiger auf den exportierten Typ, es existiert eine Erzeugungsoperation (in C++ Konstruktor)

ADO:

- es existiert nur eine 'Instanz' des ADO
- Funktionen und Prozeduren beziehen sich immer auf diese
- die Instanz wird bei Programmstart angelegt
- typische Merkmale: keine Erzeugungsoperation

(b) Anwendungsfälle: ADTs werden immer dann benötigt, wenn mehrere Objekte eines Typs erzeugt werden müssen, z.B. für die Karteikarten einer Kartei. Die Kartei selbst, sofern es nur eine gibt, könnte als ADO implementiert werden. Oft werden ADOs auch als Schnittstellen zu nur einmal vorhandenen Ressourcen wie z.B. dem Prozessor, dem Hauptspeicher, usw. genutzt.

(c) Natürlich kann kein Algorithmus zur Umwandlung eines ADO in einen ADT angegeben werden, aber man könnte grob in folgenden Schritten vorgehen.

1. Sammeln aller Konstanten und Typdefinitionen.
2. Feststellen, welche (modulglobalen) Variablen zum ADO gehören und wie sie zu initialisieren sind.
3. Erstellen der Schnittstelle der Klasse
   - in den `public`-Teil werden alle Funktionen, Prozeduren und Variablen der ADO-Schnittstelle übernommen
   - in den `private`-Teil kommen alle übrigen Funktionen, Prozeduren und Variablen.
4. In der Schnittstelle Erzeugungsoperationen (Konstruktoren) ergänzen. Zusätzlich evtl. Operationen zum Initialisieren, Kopieren und Vergleichen von Objekten.
5. Alle Variablendeklarationen werden aus der Implementierungsdatei entfernt.
6. Alle Funktions- und Prozedurdefinitionen werden angepaßt, z.B. wird
   `void push(ElementTyp element) {...}`
   zu
   `void StackTyp::push(ElementTyp element) {...}`
7. Implementieren der Erzeugungsoperationen.
8. Die Verwender des ADO sind so umzustellen, daß sie den neuen ADT verwenden. Dazu sind
   - an geeigneter Stelle Instanzen des ADT zu erzeugen und zu initialisieren
   - alle Verwendungsstellen zu ändern (`push(elem);` wird zu `stack->push(elem);`).

Probleme:

- Der ADO verwendet seinerseits weitere ADOs. Müssen auch diese jetzt als ADTs realisiert werden?
- Wenn der ADO mehrere Verwender hatte ist ggf. sicherzustellen, daß diese jetzt auf *einer* Instanz des ADT operieren.

19. Aufgabe

(a) Die internen Hilfsfunktionen konnten unverändert kopiert werden, sollten nur als `static` deklariert werden, um sie auch vor dem Linker zu verbergen. Die Schnittstellenfunktionen wurden umbenannt und haben ihren `dasHistogramm`-Parameter verloren. Dieser ist jetzt eine Modul-lokale Variable und wird von diesen Funktionen benutzt und an die internen Funktionen übergeben. Die Variable ist deklariert als

```
/* Die hier verkapselte Datenstruktur.  Die Initialisierung mit 0 macht es
   unnoetig, das Histogramm vor Gebrauch zu initialisieren. */
static HistogrammT dasHistogramm = NULL;
```

Als Beispiel für eine Schnittstellenfunktion sei hier `histodo_erhoehe` angegeben:

```
/* Erhoeht den Wert zum Schluessel, definiert ihn falls noetig.  */
void histodo_erhoeheWert(char const schluessel) {
  HistogrammEintragT* eintrag = ermittleEintrag(dasHistogramm, schluessel);
   if (NULL == eintrag)  {      // nicht da? => definieren!
      definiereSchluessel(dasHistogramm, schluessel);
      eintrag = ermittleEintrag(dasHistogramm, schluessel);
   }
   eintrag->wert++;
}
```

(b) Im Hauptprogramm muss nun kein Histogramm mehr deklariert werden. Bereits abstrahierende Funktionen wie `erstelleHistogrammAusIStream` verlieren auch nur den Histogramm-Parameter. `gebeHistogrammAus` hingegen muss nun den abstrakteren Cursor benutzen.

```
int main() {
   unsigned int anzahlWhitespace, anzahlBuchstaben;

   erstelleHistogrammAusIStream(cin);
   erstelleStatistiken(anzahlBuchstaben, anzahlWhitespace);
   gebeHistogrammAus(cout);
   gebeWertAus("Anzahl Buchstaben", anzahlBuchstaben);
   gebeWertAus("Anzahl Whitespace", anzahlWhitespace);
   return 0;
}


void erstelleHistogrammAusIStream(istream& quellStrom) {
   char gelesenesZeichen;
   while (NULL != quellStrom.get(gelesenesZeichen)) {
      histodo_erhoeheWert(gelesenesZeichen);
   }
}


void gebeHistogrammAus(ostream & zielStrom) {
   histodo_cursor * cursor;

   histodo_sortiere();

   cursor = &histodo_neuerCursor();

   if(histodo_cursorGueltig(*cursor) && histodo_cursorWert(*cursor)>0) {
      // Die groesste auszugebende Zahl
      unsigned int const MAXIMALE_HAEUFIGKEIT = histodo_cursorWert(*cursor);
      // Breite der Zahlenausgabe fuer eine huebschere Tabelle bestimmen
       unsigned int const BREITE_ANZAHL =
         (unsigned int)(log10(MAXIMALE_HAEUFIGKEIT))+1;

      cout<<"Die gelesenen Zeichen und ihre Anzahlen, "
          <<"in absteigender Reihenfolge:" <<endl;
      while (histodo_cursorGueltig(*cursor)) {
         zielStrom << "`";
         if (isprint(histodo_cursorSchluessel(*cursor)) ) {
            zielStrom << histodo_cursorSchluessel(*cursor);
         } else {
            // Das Zeichen wuerde gedruckt schlecht aussehen (etwa ein
            // Zeilenumbruch, Tabulator oder BELL), also drucke nur seine
            // Nummer.
            zielStrom << "<" << (unsigned int)histodo_cursorSchluessel(*cursor)
                      << ">";
         }
         /// Mit setw wird die Breite des Feldes definiert, in dem die Zahl
         /// ausgegeben wird.  Die Ausgabe-Bibliothek fuellt das Feld von
         /// links mit Leerzeichen auf, so dass die Werte rechtsbuendig
         /// untereinander stehen.
         zielStrom << "': " <<setw(BREITE_ANZAHL) <<histodo_cursorWert(*cursor)
                   << endl;
         histodo_cursorWeiter(*cursor);
      }
   } else {
      zielStrom << "Keine Zeichen gezaehlt." << endl;
   }
}
```