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

Aachen, 08. 06. 2007

Nagl, Ranger, Wörzberger

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

— Blatt 8 —

17. Aufgabe *Getrennte Übersetzung*: *(4 Punkte)*

Um bei großen Programmen den Übersetzungsaufwand bei Änderungen zu verringern, ist es möglich, das Programm auf mehrere Dateien zu verteilen, die unabhängig voneinander zu Objektdateien übersetzt werden. In einem zweiten Schritt (*Linken*) werden dann die Objektdateien zu einem ausführbaren Programm zusammengebunden. Bei einer Änderung sind dann nur die geänderten Programmdateien neu zu übersetzen.

In dieser Aufgabe soll am Beispiel eines einfachen `Hello, World!`-Programms geübt werden, wie man ein Programm auf mehrere Übersetzungseinheiten (Objektdateien) verteilt und diese zusammenbindet.

(a) Unser Programm soll ein Funktionsmodul (`message`) verwenden, das eine Funktion exportiert, die den auszugebenden Text (z.B. `Hello, World!`) liefert. Ein zweites Funktionsmodul (`helloio`) dient der Ausgabe eines Textes (z.B. auf dem Bildschirm). Beide Module werden vom Hauptteil des Programms (`hello`) benutzt, um `Hello, World!` auszugeben.

Erstellen Sie Schnittstellen und Rümpfe für beide Module und den Hauptteil des Programms. Verwenden Sie 5 Dateien mit folgenden Dateinamen: `message.h`, `message.cc`, `helloio.h`, `helloio.cc` und `hello.cc`. *(3 Punkte)*

(b) Übersetzen Sie die `*.cc`-Dateien einzeln und Linken Sie die entstehenden Objektdateien (`message.o`, `helloio.o` und `hello.o`) zu einem ausführbaren Programm. Verwenden Sie beim Übersetzen die Option `-c` des `g++` (oder eine entsprechende Option Ihres Compilers), damit nicht gleich vom `g++` der Linker gestartet wird. Aus den entstehenden Objektdateien kann mit `g++ message.o helloio.o hello.o -o hello` das ausführbare Programm `hello` gelinkt werden. Dokumentieren Sie das Ergebnis ihrer Schritte durch die Ausgabe des Verzeichnisinhaltes und den Befehlen, die sie eingegeben haben. *(1 Punkt)*

18. Aufgabe *Modularten ADO und ADT*: *(6 Punkte)*

(a) In der Vorlesung wurden die beiden Modularten *Abstraktes Datenobjektmodul* (ADO) und *Abstraktes Datentypmodul* (ADT, in C++ meist als Klasse realisiert) vorgestellt.

Beschreiben Sie stichpunktartig die Eigenschaften beider Modularten. Welche typischen Merkmale haben Schnittstellen von ADOs bzw. ADTs? *(1 Punkt)*

(b) Für welche Anwendungsfälle würden Sie ADTs verwenden, für welche ADOs? *(2 Punkte)*

(c) Ein älteres, in C geschriebenes Programm soll überarbeitet und nach C++ portiert werden. Dabei sollen einige der bisher als ADOs realisierten Module zu Klassen (ADTs) werden.

Überlegen Sie sich möglichst allgemein, in welchen Schritten Sie dabei vorgehen würden und welche Probleme auftauchen können. Welche Änderungen sind im Modul nötig, welche beim Verwender? *(3 Punkte)*

19. Aufgabe *Histogramm als ADO auslagern*: *(10 Punkte)*

Im Histogramm-Programm sollen jetzt Module eingeführt werden. Damit wird im Hauptprogramm ein weiterer Abstraktionsschritt nötig: Funktionen, die noch über die Zeigerstruktur iterieren, müssen jetzt auch auf eine Schnittstelle zugreifen.

(a) Implementieren Sie den Rumpf des Moduls `histodo`, dessen Schnittstelle unten angegeben ist. Versuchen Sie, möglichst viel Code aus der Lösung der letzten Aufgabe zu kopieren. *(5 Punkte)*

(b) Passen Sie das Hauptprogramm an, um den neuen Modul zu verwenden. *(5 Punkte)*

```
/* Oeffentliche Schnittstelle des Histogramm-Datenobjekts.
   Das Histogramm speichert zu jedem 'char'-Schluessel einen 'unsigned
   int'-Wert.  Der Wert kann erhoeht und ausgelesen werden.  Nach der
   Initialisierung sind saemtliche Werte 0.

   Alle hier definierten Namen beginnen mit 'histodo_', um die
   Zugehoerigkeit zu kennzeichnen.*/
#ifndef __INC_histodo_h__
#define __INC_histodo_h__
/* Setzt alle Zaehler im Histogramm auf 0. */
void histodo_initialisiere();
/* Erhoeht den Wert zum Schluessel.  */
void histodo_erhoeheWert(char const schluessel);
/* Liefert den Wert zum Schluessel. */
unsigned int histodo_ermittleWert(char const schluessel);

/* Um ueber alle interessanten (Wert>0) Schluessel des Histogramms zu
   iterieren, wird hier ein Cursor-Typ definiert.  Dieser abstrahiert
   vollstaendig von der internen Speicherung.

   Diese Cursor erlauben nur Leseoperationen und verlassen sich darauf, dass
   waehrend eines Durchlaufs das Histogramm nicht geaendert wird.
*/
class histodo_cursor;

/* Liefert einen neuen Cursor. */
histodo_cursor& histodo_neuerCursor();

/* Testet, ob der Cursor noch gueltig ist.  */
bool histodo_cursorGueltig(histodo_cursor const & cursor);

/* Verschiebt den Cursor auf einen weiteren Eintrag mit Wert>0 und noch
   nicht besuchtem Schluessel.  Sind bereits alle Schluessel besucht, wird
   der Cursor stattdessen ungueltig.  */
void histodo_cursorWeiter(histodo_cursor& cursor);

/* Liefert den Schluessel des Eintrags, auf den der Cursor zeigt, falls
   der Cursor gueltig ist. */
char histodo_cursorSchluessel(histodo_cursor const & cursor);
```

```
/* Liefert den Wert des Eintrags, auf den der Cursor zeigt, falls
   der Cursor gueltig ist. */
unsigned int histodo_cursorWert(histodo_cursor const & cursor);

/* Sortiere die Eintraege im Histogramm in absteigender Reihenfolge.
   Nachfolgende Cursor-Duchlaeufe erhalten die Eintraege dann in dieser
   bestimmten Abfolge.
*/
void histodo_sortiere();
#endif
```

Die Klasse `histodo_cursor` ist im Rumpf von `histodo` wie folgt implementiert:

```
/* Da diese Funktionen sehr genau dem entsprechen, was in der Schnittstelle
   angeboten wird, sind sie hier nicht noch einmal kommentiert.
*/
class histodo_cursor
{
public:
   histodo_cursor(HistogrammT dasHistogramm) { eintrag = dasHistogramm; }

   bool gueltig() const { return eintrag != NULL; }

   unsigned int wert   () const {
      if(gueltig()) { return eintrag->wert; }
                else {
                       cerr << "Ungueltiger Eintrag!\n";
                       return 0;
                     }
   }
   char  schluessel () const {
      if(gueltig()) { return eintrag->schluessel; }
                else {
                       cerr << "Ungueltiger Eintrag!\n";
                       return 0;
                     }
   }
   void weiter() {
      if(gueltig()) { eintrag = eintrag->naechster; }
                else { cerr << "Ungueltiger Eintrag!\n"; }
   }
private:
   HistogrammEintragT const * eintrag;
};
```

Als Beispiel für die Verwendung des Cursor ist hier der Rumpf von `erstelleStatistiken` angegeben:

```
   histodo_cursor* cursor = &histodo_neuerCursor();
   while (histodo_cursorGueltig(*cursor)) {
       if ( isalpha(histodo_cursorSchluessel(*cursor)) ) {
         anzahlBuchstaben += histodo_cursorWert(*cursor);
      }
      if ( isspace(histodo_cursorSchluessel(*cursor)) ) {
         anzahlWhitespace += histodo_cursorWert(*cursor);
      }
      histodo_cursorWeiter(*cursor);
   }
```

Abgabe: 15. 06. 2007 vor der Vorlesung.