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

Aachen, 12. 05. 2006

Nagl, Becker, Ranger, Wörzberger

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

— Blatt 5 —

10. Aufgabe *Zahlen raten*: *(8 Punkte)*

In dieser Aufgabe geht es um ein einfaches Spiel, bei dem sich ein Spieler eine Zahl innerhalb eines bestimmten Intervalls (z. B. 0..99) merkt und der zweite Spieler versucht, die gedachte Zahl mit möglichst wenigen Versuchen zu erraten. Spieler 1 antwortet auf einen Rateversuch jeweils mit „richtig“, „größer“ oder „kleiner“. Der Spieler, der weniger Versuche benötigt, hat gewonnen.

(a) Die einfachste Strategie besteht darin, alle möglichen Zahlen von 0 bis 99 der Reihe nach zu raten, bis man die richtige gefunden hat. Wieviele Versuche sind im Schnitt nötig, um eine Zahl zu raten, wieviele im schlechtesten Fall? *(2 Punkte)*

(b) Überlegen Sie sich ein besseres Verfahren und beschreiben Sie es umgangssprachlich oder besser in Form von Pseudocode. *(4 Punkte)*

(c) Bestimmen Sie die Anzahl der Versuche, die im schlechtesten Fall mit Ihrem Verfahren nötig sind. *(2 Punkte)*

11. Aufgabe *Zeiger und Listen*: *(12 Punkte)*

Das Histogramm-Programm benutzt ein Feld, um die Zähler der vorgekommenen Buchstaben zu verwalten. Damit war im Programm die Gesamtzahl möglicher Zeichen hart codiert. Der „Trick“, den Code eines Zeichens als Index im Zählerfeld zu benutzen, machte das Programm auch nicht unbedingt verständlicher. Dies wollen wir nun mit einer verketteten Liste von Schlüssel/Wert-Paaren verbessern.

Die neue Histogramm-Datenstruktur besteht aus Schlüsseln und Werten (auch bezeichnet als *Tabelle* oder englisch *map*). Dazu gibt es (öffentliche) Funktionen, die den Wert zu einem Schlüssel liefern, den Wert erhöhen oder alle Werte auf 0 zurücksetzen. Aus dieser Sicht gibt es keine „unbekannten Schlüssel“, sondern nur Zähler, die bei ungezählten Schlüsseln eben 0 sind. Intern wird die Tabelle als verkettete Liste abgespeichert. Hierzu gibt es Funktionen, die Einträge abfragen oder einfügen.

Einige der Funktionen sind bereits ausformuliert, in anderen fehlen Teile des Codes, gekennzeichnet mit `/* TODO: Implementierung erstellen*/`. **Ergänzen sie die fehlenden Codestücke!**

Aus Platzgründen soll an dieser Stelle nicht das komplette Programm wiedergegeben werden, sondern nur die wichtigsten Häppchen. Den gesamten Rest finden Sie auf der Vorlesungshomepage bei den Uebungsaufgaben `http://www-i3.Informatik.RWTH-Aachen.de/ggdi`.

Die Listen-Datenstruktur ist wie folgt definiert:

```
struct HistogrammEintragT {       // Ein Listenelement mit Verkettung
   char               schluessel; // Schluessel, nach dem gesucht wird
   unsigned int       wert;       // Wert, der zum Schluessel abgelegt wird
   HistogrammEintragT *naechster;  // Verkettung mit dem naechsten Eintrag
};
typedef HistogrammEintragT *HistogrammT; // Ein gesamtes Histogramm
```

Diese Funktion sucht nach dem internen Eintrag zu einem Schlüssel:

```
/* Liefert aus einem Histogramm zu einem Schluessel einen Zeiger auf den
   Eintrag (falls bekannt) oder NULL (falls nicht).  Intern. */
HistogrammEintragT *ermittleEintrag(HistogrammT dasHistogramm,
                                    char const  schluessel) {
   HistogrammEintragT *cursor   = dasHistogramm;
   HistogrammEintragT *eintrag  = NULL;  // Zwischenspeicher fuer Ergebniszeiger
   bool gefunden = false; // Flag: Eintrag zum Schluessel gefunden?
   while (NULL != cursor && !gefunden) {
      if (schluessel == cursor->schluessel) { // Eintrag zum Schluessel gefunden!
         eintrag  = cursor;     // Ergebnis merken
         gefunden = true;       // Schleife kann beendet werden
      } else {
         cursor = cursor->naechster; // Cursor auf den naechsten Eintrag
      }
   }
   return eintrag;              // NULL, falls nicht gefunden
}
```

(a) Ergänzen Sie den hier fehlenden Test. Die Hauptarbeit erledigt eine bereits bekannte Funktion, die Lösung ist *sehr* kurz. *(2 Punkte)*

```
/* Ermittelt, ob ein Zeichen bereits im Histogramm vorkommt.  Intern. */
bool schluesselBekannt(HistogrammT dasHistogramm, char const schluessel) {
/* TODO: Implementierung erstellen */
}
```

(b) In dieser Funktion fehlen die Anweisungen, die den neuen Eintrag im Histogramm einbetten. Eine einfache Lösung kommt mit zwei Zuweisungen aus; mehr ist hier auch nicht verlangt. *(2 Punkte)*

```
/* Legt einen Eintrag fuer den Schluessel im Histogramm an, mit Wert 0.
   Nach definiereSchluessel(h,s) liefert ermittleEintrag(h,s) nicht NULL.
   Intern. */
void definiereSchluessel(HistogrammT& dasHistogramm, char const schluessel) {
   if (!schluesselBekannt(dasHistogramm, schluessel)) {
      HistogrammEintragT* neuerEintrag = new HistogrammEintragT;
      neuerEintrag->schluessel = schluessel;
      neuerEintrag->wert       = 0;
/* TODO: Implementierung erstellen */
   } else { cout << "Schluessel bekannt." << endl; }
}
```

(c) Formulieren Sie den Rumpf aus. *(4 Punkte)*

```
/* Liefert den Wert zum Schluessel.  Oeffentlich. */
unsigned int ermittleWert(HistogrammT dasHistogramm, char const schluessel) {
/* TODO: Implementierung erstellen */
}
```

(d) Formulieren Sie die Iteration über alle bekannten Schlüssel, um die gleichen Statistiken wie in Aufgabe 7 zu erstellen. *(4 Punkte)*

```
/* Einige einfache Statistiken.  */
void erstelleStatistiken(HistogrammT dasHistogramm,
                         unsigned int &anzahlBuchstaben,
                         unsigned int &anzahlWhitespace) {
   anzahlBuchstaben = 0;
   anzahlWhitespace = 0;
   HistogrammEintragT const * cursor = dasHistogramm;
/* TODO: Implementierung erstellen, Tip: haeufigstesZeichen*/
}
```

Nun noch einige wichtige Funktionen:

Der Eintrag zu einem Schlüssel kann benutzt werden, um den Wert zu ändern.

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

Bei der Benutzung des Histogramms hat sich nur der Zugriff geändert.

```
/* Zaehlt die Zeichen des Eingabestroms. */
void erstelleHistogrammAusIStream(HistogrammT& zielHistogramm,
                                  istream&     quellStrom) {
    char gelesenesZeichen;

    quellStrom.get(gelesenesZeichen);

    while (!quellStrom.eof()) {
erhoeheWert(zielHistogramm, gelesenesZeichen);
quellStrom.get(gelesenesZeichen);
    }
}
```

Die Schleife über alle Elemente ist schon stärker umgebaut:

```
/* Ermittelt ein haeufigstes Zeichen (nicht eindeutig). */
char haeufigstesZeichen(HistogrammT dasHistogramm) {
   unsigned int maximaleHaeufigkeit  = 0;       // Zahl 0
   char         haeufigstesZeichen   = '\x0';   // Zeichen mit Code 0
   HistogrammEintragT const * cursor = dasHistogramm;
   while (NULL != cursor) {       // Schleife ueber Eintraege
      if (cursor->wert > maximaleHaeufigkeit) { // neues Maximum
         haeufigstesZeichen  = cursor->schluessel;
         maximaleHaeufigkeit = cursor->wert;
      }
      cursor = cursor->naechster; // Cursor auf den naechsten Eintrag
   }
   return haeufigstesZeichen;     // nur 0, falls Histogramm leer
}
```

Das Hauptprogramm ist weitgehend unverändert:

```
/* Hauptprogramm */
int main() {
   HistogrammT  gezaehlteZeichen = NULL;
   unsigned int anzahlWhitespace, anzahlBuchstaben;

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

Abgabe: 19. 05. 2006 vor der Vorlesung.