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

Aachen, 19. 05. 2006

Nagl, Becker, Ranger, Wörzberger

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

— Blatt 6 —

12. Aufgabe *Programm Zahlenraten*: *(10 Punkte)*

Das Spiel „Zahlenraten“ vom letzten Aufgabenblatt soll jetzt schrittweise implementiert und verbessert werden. Zur Suche nach der vom Spieler gedachten Zahl soll das von Ihnen zu implementierende Programm folgenden, als Pseudocode angegebenen, Algorithmus verwenden.

```
Untergrenze eingeben
Obergrenze eingeben

Suche nach der geratenen Zahl initialisieren
solange Zahl noch nicht geraten
    // Zahl aus Intervall auswählen
    gerateneZahl = (aktuelleUntergrenze + aktuelleObergrenze) / 2
    wenn gerateneZahl richtig war: Fertig

    // neues Intervall bestimmen
    wenn gerateneZahl zu groß
        aktuelleObergrenze = gerateneZahl-1
    wenn gerateneZahl zu klein
        aktuelleUntergrenze = gerateneZahl+1
```

Bevor Sie mit der Lösung von Aufgabenteil a) beginnen, lesen Sie sich auch die anderen Aufgabenteile durch, um einen Überblick über die zur Erstellung des kompletten Programms nötigen Teilschritte zu gewinnen. Nach jedem Teilschritt sollte sich das Programm übersetzen lassen.

(a) Implementieren und Testen Sie zunächst den Rahmen des Programms, also die Eingabe der Unter- und Obergrenzen durch den Benutzer. Zusätzlich soll das Programm dem Benutzer die eingegebenen Grenzen noch einmal anzeigen und ihn dazu auffordern, sich eine Zahl zu denken. *(2 Punkte)*

(b) Jetzt wird der Algorithmus implementiert, der eine Zahl berechnet und den Benutzer danach fragt, ob die geratene Zahl richtig, kleiner oder größer ist (Eingabe von `r` `k`, `g`). Sollte Ihnen der angegebene Algorithmus noch nicht klar sein, spielen Sie ihn an einem kleinen Beispiel durch oder versuchen Sie die iterative Fassung der binären Suche (im Skript auf Seite 3–8) zu verstehen. Beachten Sie jedoch, dass die binäre Suche in diesem Fall immer funktioniert, da das gesuchte Element in jedem Fall existiert. In der Regel wird dies jedoch nicht der Fall sein.

Ihr Programm sollte jetzt bereits eine Zahl raten können. *(5 Punkte)*

(c) Damit das Programm wirklich benutzbar ist, dürfen Fehleingaben nicht zu undefinierten Zuständen, Endlosschleifen, Abstürzen o.ä. führen. Erweitern Sie das Programm so, daß es Fehleingaben erkennt, dem Benutzer eine Fehlermeldung ausgibt und ihm ggf. eine Korrektur der Fehleingabe ermöglicht. Erkannt werden sollen insbesondere

- falsche Intervallgrenzen,
- falsche Eingabe von `r`, `k`, `g` für richtig, kleiner oder größer und
- Benutzer lügt bei der Eingabe von `k`(leiner) oder `g`(rößer).

Es ist an dieser Selle jedoch nicht notwendig, daß Sie die Routine zum Einlesen von Zahlen (`unsigned grenze; cin » grenze;`) durch eine eigene ersetzen. *(3 Punkte)*

13. Aufgabe *Programmierstil, Rückgabewerte*: *(10 Punkte)*

Beim Programmieren hat man oft mehrere funktionierende Möglichkeiten, die sich aber bezüglich Lesbarkeit und Wartbarkeit unterscheiden. Je nach Einsatzzweck können verschiedene Möglichkeiten verschieden gut geeignet sein; *die* beste Lösung gibt es normalerweise nicht.

Am Beispiel der aus der Vorlesung bekannten Funktion `linSuche()` sollen hier verschiedene Möglichkeiten zur Rückgabe von Werten aus Funktionen bzw. Prozeduren verglichen werden.

1. Wie in der Vorlesung angegeben: Zustand (gefunden oder nicht gefunden) und Index eines gefundenen Elements werden beide über Referenzparameter zurückgegeben.

```
/*Typdefinitionen, ... */
void linSuche(it_FT &f, item ges_Wert, index ia, index ie,
              index &ind,                    // Index falls gefunden
              zust &z) {                     // Zustand: gef oder nicht_gef

  /* Suchschleife, vgl. Skript Folie 3-5*/
  index indx; index og=ie+1;
  indx=og;
  for (index i=ia; i<og; i++) {
    if (f[i]==ges_Wert) {
     indx=i; break;
  }
  if (indx < og) {
    ind = indx; z = gef;                     // Rueckgabewerte setzen
  } else {
    z = nicht_gef;
  }
}
```

2. Alternative Implementierung, bei der beide zurückgegebenen Werte (`z` und `ind` in einer `struct` zusammengefaßt sind.)

```
/* Typdefinitionen, ... */
struct Suchergebnis {
  zust  z;
  index ind;
}

Suchergebnis linSuche(it_FT &f, item ges_Wert, index ia, index ie) {
  Suchergebnis rueckgabe;
  /* Suchschleife */
  if (indx < og) {
    rueckgabe.z = gef;
    rueckgabe.ind = indx;
  } else {
    rueckgabe.z = nicht_gef;   // rueckgabe.ind bleibt undefiniert
  }
  return rueckgabe;
}
```

(a) Erstellen Sie eine dritte Variante. *(3 Punkte)*

(b) Geben Sie je einen Vor- und einen Nachteil für jede der Lösungen an. *(3 Punkte)*

(c) Bei der im Skript angegebenen Prozedur zur linearen Suche wird zwischen der Suche nach dem Wert und der Entscheidung unterschieden, ob ein Wert gefunden wurde. Geben Sie eine Lösung an, bei der der gefundene Wert direkt aus der Schleife zurückgegeben wird.

Die Bestimmung der Rückgabewerte (`if (index < og) ...`) in der Funktion `linSuche()` läßt sich auch eleganter formulieren. Erläutern Sie, welche Nachteile diese Vorgehensweise mit sich bringt.

(4 Punkte)

Abgabe: 26. 05. 2006 vor der Vorlesung.