Praktikum zur Vorlesung
GRUNDLAGEN DER INFORMATIK I
Studiengang Elektrotechnik
WS 02/03

AG Betriebssysteme FB3
Kirsten Berkenkötter

Ausgabe: 21./28.1.2003

Praktikum 6: Verwendung von Bibliotheksfunktionen

In der Bibliothek <*stdlib.h*> sind u.a. die Funktionen *srand()* und *rand()* definiert, mit deren Hilfe man Zufallszahlen erzeugen kann:

```
void srand(unsigned int startwert);   //Zufallszahlengenerator initialisieren
int rand(void); //Zufallszahlen erzeugen
```

Die erste Funktion muss einmal aufgerufen werden, um den Zufallszahlengenerator zu starten. Üblicherweise greift man für den Startwert auf die Funktion *time()* zurück, die in der Headerdatei <*time.h*> definiert ist und die Anzahl der Sekunden seit dem 1.1.1970 zuückgibt:

```
srand(time(NULL));
```

Danach können mit der zweiten Funktion Zufallszahlen abgerufen werden, die im Bereich 0..32767 liegen.

Weiterhin sind in der Bibiliothek <*stdlib.h*> die Funktionen *qsort()* und *bseach()* definiert, welche die Algorithmen Quicksort, bzw. binäre Suche implementieren:

```
void qsort(void *array, size_t anzahl, size_t groesse,
      int (*vergl_funktion) (const void *, const void *));
void *bsearch (const void *such_zeiger,
      const void *array, size_t anzahl, size_t, groesse,
      int (*vergl_funktion)(const void *, const void *));
```

Dabei bedeuten die Parameter folgendes:

```
void *array: Zeiger auf das zu sortierende Array, bzw. Teilarray
size_t anzahl: Anzahl der Elemente im Array, bzw. Teilarray
size_t groesse: Grösse eines Elements im Array
int (*vergl_funktion)(const void *, const *):
      Zeiger auf eine Vergleichsfunktion mit dem Rückgabewert int,
      die zwei Parameter vom Typ void-Zeiger erwartet
```

1 Aufgabe 1: Zufallszahlen, Sortieren und Suchen

In dieser Aufgabe soll die Verwendung der Bibliotheksfunktionen *srand()*, *rand()*, *qsort()* und *bsearch* der Bibliothek <*stdlib.h*> geübt werden.

Legt im Hauptprogramm ein *int*-Array der Größe 10 an, dass ihr dann mit Zufallszahlen im Bereich von 1..100 initialisiert und gebt es aus. Mit Hilfe von *qsort()* sollen diese Zahlen dann sortiert und nochmals ausgegeben werden. Danach soll eine neue Zufallszahl erzeugt und im vorsortierten Array gesucht werden. Dies soll mit der Funktion *bsearch()* geschehen. Gebt aus, welche Zahl ihr gesucht habt und ob sie gefunden/nicht gefunden wurde.

Die Vergleichsfunktion, die für die Funktionen *qsort()* und *bsearch*() benötigt wird, muss also int-Zahlen vergleichen, d.h. Rückgabewert -1, falls zahl1 < zahl2, Rückgabewert 0, falls zahl1 == zahl2 und Rückgabewert 1, falls zahl1 > zahl2.

2 Aufgabe 2: Zeichenketten, Sortieren und Suchen

In dieser Aufgabe soll die Verwendung von *qsort()* und *bsearch()* für Zeichenketten geübt werden.

Legt im Hauptprogramm ein Array der Größe 10 an, dass jeweils Zeichenketten der Größe 20 aufnehmen kann. Die 10 Zeichenketten sollen mit Hilfe der Funktion *scanf()* eingelesen (z.B. Namen) und dann mit *qsort()* sortiert werden. Gebt das Array vor und nach dem Sortieren aus. Danach lest mit *scanf* wiederum einen Namen ein, den ihr dann mit *bsearch()* in dem Array sucht. Gebt an, welchen Namen ihr gesucht habt und ob er gefunden/nicht gefunden wurde.

Die Vergleichsfunktion, die für die Funktionen *qsort()* und *bsearch*() benötigt wird, muss also Zeichenketten vergleichen, d.h. Rückgabewert -1, falls kette1 < kette2, Rückgabewert 0, falls kette1 == kette2 und Rückgabewert 1, falls kette1 > kette2. Für die Vergleiche wird die normale alphabetische Sortierung zugrunde gelegt, d.h. Großbuchstaben sind größer als Kleinbuchstaben und a > b > c ... > z.