PT Übung – 1. Semester

Übung 2

Ziele:

- Verwenden von Schleifen
- Entscheidungen mit if-then-else

Übungsaufgaben:

1. Schreiben Sie ein Programm, welches 10 Noten eines Studenten einliest und den Notendurchschnitt und die Anzahl der „Nicht Genügend“ ausgibt.

   Lesen Sie dazu die 10 Noten (1-5) in einer Schleife von der Tastatur ein, bilden Sie fortlaufend die Summe der Noten und zählen Sie die „Nicht Genügend“. Berechnen Sie anschließend den Notendurchschnitt und geben Sie die Ergebnisse am Bildschirm aus.

2. Schreiben Sie ein Programm zum Erraten von Zufallszahlen im Bereich 1-100.

   Erzeugen Sie zuerst eine Zufallszahl mit Hilfe des folgenden Codes (inkludieren Sie zur Verwendung der Funktion time() die Datei „time.h“):

   ```
   srand(time(NULL));                  /* Initialisierung */
   rand();  /* Die erste Zufallszahl ist meist ungeeignet */
   zufallszahl = 1+(int)(100.0*rand()/(RAND_MAX+1.0));
   ```

   Lesen Sie anschließend Zahlen in einer Schleife von der Tastatur ein und geben Sie einen Hinweis aus, ob die Zufallszahl größer oder kleiner ist. Beenden Sie die Schleife entweder nach 10 Versuchen oder bei Erraten der Zahl ab.

Zusatzaufgaben:

1. Ändern Sie das Beispiel 2 so ab, dass der Benutzer anstatt einer Zahl zwischen 1 und 100 einen Buchstaben zwischen „a“ und „z“ erraten muss.

Kontrollfragen:

- Wie brechen Sie eine Schleife ab?
- Ist es sinnvoll Variablen vom Typ float auf Gleichheit zu überprüfen?

Anmerkung: Die Funktion `srand(int seed)` dient der Initialisierung des Pseudozufallszahlengenerator. Abhängig von dem übergebenen Wert wird eine Folge von Zufallszahlen generiert. Übergibt man jedes Mal die gleiche Zahl an die Funktion `srand()`, so erhält man immer die selbe Folge von Zufallszahlen (z.B.: 1, 97, 5, 13, 45, 43,…). Um immer eine andere Folge von Zufallszahlen zu erhalten, hat es sich etabliert die Systemzeit (in Sekunden seit 1.1.1970 --- `time()`) an die Funktion `srand()` zu übergeben, wodurch sichergestellt ist, dass man immer eine andere Zufallszahlenfolge erhält. Leider ist es bei manchen Zufallszahlengeneratoren der Fall, dass die erste Zufallszahl häufig dieselbe ist. Daher ist es günstig erst mit der zweiten Zufallszahl zu beginnen (durch ignorieren der ersten --- erster `rand()` Befehl).

```
RAND(3)            Linux Programmer's Manual            RAND(3)

NAME
       rand, srand - random number generator.

SYNOPSIS
       #include <stdlib.h>

       int rand(void);

       void srand(unsigned int seed);

DESCRIPTION
       The  rand()  function  returns  a  pseudo-random  integer
       between 0 and RAND_MAX.

       The srand() function sets its argument as the seed  for  a
       new  sequence  of pseudo-random integers to be returned by
       rand().  These sequences are repeatable by calling srand()
       with the same seed value.

       If no seed value is provided, the rand() function is auto-
       matically seeded with a value of 1.

RETURN VALUE
       The  rand()  function  returns  a  value  between  0   and
       RAND_MAX.  The srand() returns no value.

NOTES
       The  versions of rand() and srand() in the Linux C Library
       use the same random number generator as random() and sran-
       dom(),  so the lower-order bits should be as random as the
       higher-order bits.  However, on older  rand()  implementa-
       tions,  the lower-order bits are much less random than the
       higher-order bits.
```

In *Numerical Recipes in C: The Art of Scientific Computing* (William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling; New York: Cambridge University Press, 1990 (1st ed, p. 207)), the following comments are made:

> "If you want to generate a random integer between 1 and 10, you should always do it by
>
> ```
> j=1+(int) (10.0*rand()/(RAND_MAX+1.0));
> ```
>
> and never by anything resembling
>
> ```
> j=1+((int) (1000000.0*rand()) % 10);
> ```
>
> (which uses lower-order bits)."

Random-number generation is a complex topic. The *Numerical Recipes in C* book (see reference above) provides an excellent discussion of practical random-number generation issues in Chapter 7 (Random Numbers).

For a more theoretical discussion which also covers many practical issues in depth, please see Chapter 3 (Random Numbers) in Donald E. Knuth's *The Art of Computer Programming*, volume 2 (Seminumerical Algorithms), 2nd ed.; Reading, Massachusetts: Addison-Wesley Publishing Company, 1981.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

random(3), **srandom**(3), **initstate**(3), **setstate**(3)