PR3 - Leseaufgabe zur Vorlesung 6

(Stand 2021-10-01 14:49)

Prof. Dr. Holger Peine
Hochschule Hannover
Fakultät IV – Abteilung Informatik
Raum 334, Tel. 0511-9296-1830
Holger.Peine@hs-hannover.de

L.6 Funktionen

L.6.1 Zufallszahlen

Mit folgenden Funktionen lassen sich in C Zufallszahlen erzeugen:

- Die Funktion `int rand(void)` in `stdlib.h` erzeugt eine Zufallszahl zwischen 0 und `RAND_MAX`.
- Die Funktion void `srand(unsigned int)` initialisiert den Zufallszahlengenerator.
  - Der Parameter bestimmt nach einem unspezifizierten Algorithmus, welche Zufallszahl die erste ist.
  - Häufig: Initialisierung mit der Uhrzeit als Parameter, damit verschiedene Programmaufrufe immer andere Zufallszahlen liefern (Achtung: Für Zufallszahlen, die geheim sein müssen, wäre das nicht sicher genug!)

Mit diesen Funktionen kann man eine komfortablere Funktion implementieren, die eine ganze Zufallszahl im Bereich {0,1,...,hi-1} berechnet:

```
#include <stdlib.h>
#include <time.h>
int randomNumber(int hi)  /* Rückgabe in {0,1,...,hi-1} */
{
    const double scale = rand()/((double)RAND_MAX+1.0);
    int i = (int)(scale*hi);
    /* falls wegen Rundungsfehlern der gewünschte Wertebereich
       ueberschritten wurde: eingrenzen */
    return (i >= hi ? hi - 1 : i);
}
...
int main(void) {
    srand( (unsigned)time(NULL) );
    ...
}
```

L.6.2 errno, strerror

In der Headerdatei `errno.h` ist eine globale Variable `errno` vom Typ `int` deklariert (oder eine entsprechende Präprozessor-Definition; `errno` kann jedenfalls wie ein `int` -Wert gelesen werden). Diese zeigt bei einigen Bibliotheksfunktionen an, welche Art von Fehler aufgetreten ist. Der Wert der Variable `errno` kann z. B. den Wert `EDOM` annehmen, der ein unzulässiges Argument für eine mathematische Funktion meldet (deklariert in `errno.h`).

Mit `errno` können Bibliotheksfunktionen genauere Informationen über die Ursache eines Fehlers anzeigen, als es oft über den Rückgabewert der Funktion allein möglich wäre. Anwendung z. B. wie folgt:

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <math.h>

int main(void) {
  double x, y = -5;

  errno = 0; /* Vorher initialisieren */
  x = sqrt(y);
  if (errno != 0) {
    printf("sqrt meldet den Fehlercode %d (%s)\n", errno, strerror(errno));
  }
  return 0;
}
```

Ausgabe:
```
sqrt meldet den Fehlercode 33 (Numerical argument out of domain)
```

Die in `string.h` deklarierte Funktion `strerror` liefert eine Fehlermeldung zum übergebenen Fehlercode.

L.6.3 Umwandlung von Zeichenketten in Zahlen

Gegeben sei eine Zeichenkette:
```
char[] text = "779";
```

Wie lässt sich die Zeichenkette in eine Zahl umwandeln?

L.6.3.1 atoi

Die Funktion `int atoi(const char* string)` wandelt eine Zeichenkette in eine Zahl um. Das Problem bei dieser alten und eigentlich nur aus Kompatibilitätsgründen noch vorhandenen Funktion ist, dass ihr Verhalten im Fehlerfall undefiniert ist.

Ähnliche (alte) Funktionen gibt es für die Umwandlung in Fließkommazahlen (`atof`) und in `long int` (`atol`).

L.6.3.2 sscanf

Die Funktion `int sscanf(const char* s, const char* format, ...)` liest aus der Zeichenkette `s` einen Text aus und versucht ihn gemäß der Formatangabe `format` umzuwandeln. Die weiteren Parameter (…) sind Zeiger auf Variable, die zu den Formatangaben passen, und in denen im Erfolgsfall die aus `s` gelesenen Werte gespeichert werden.

Die Rückgabe ist entweder `EOF` (definiert in `stdio.h`) oder die Anzahl der erfolgreich eingelesenen Variablen.

Beispiel:
```
  char text[]= "779";
  int zahl;
  if (sscanf(text, "%d", &zahl) != 1) {
    printf("Kann '%s' nicht in Zahl umwandeln.\n", text);
  } else {
    printf("%d\n", zahl);
  }
```

Aber auch bei `sscanf` gibt es ein Problem. Wenn der Inhalt von `text` z. B. „779x“ ist, wird dennoch die Zahl 779 konvertiert. Das Zeichen „x“ wird als erstes Zeichen des nächsten zu konvertierenden Objekts interpretiert; falls `format` aber kein weiteres `%...` enthält, wird das "x" ignoriert.

L.6.3.3 strtol

Die Funktion `long strtol(const char* s, char** endptr, int base)` bietet eine noch bessere Fehlerkontrolle als `sscanf`.

Diese Funktion zerlegt den gegebenen String `s` in drei Teile:

- zunächst eine optionale Folge von Whitespaces
- dann einen Teil, der in der gegebenen Basis `base` als Zahl interpretiert werden kann
- und schließlich den Rest des Strings inkl. des 0-Terminators.

Dann versucht die Funktion den mittleren Teil in eine ganze Zahl zu konvertieren und liefert das Ergebnis als Rückgabe zurück. Wenn keine Konvertierung stattfinden konnte, wird 0 zurück gegeben. Im Zeiger `*endptr` legt `strtol` einen Zeiger auf den Anfang des dritten Teils ab (der auch nur aus dem 0-Terminator bestehen kann).

Eine Besonderheit ist zu beachten, wenn der konvertierte Wert zu groß oder zu klein ist (größer als `LONG_MAX` oder kleiner als `LONG_MIN`, beide deklariert in `limits.h`). In diesem Fall wird `LONG_MAX` bzw. `LONG_MIN` zurück gegeben und nebenbei die globale Variable `errno` auf den Wert `ERANGE` gesetzt (beide deklariert in `errno.h`).

Beispiel:
```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

int main(void) {
  char text[] = "779";
  long zahl;
  char* endptr = NULL;
  errno = 0; /* Vorher initialisieren */
  zahl = strtol(text, &endptr, 10);  /* 10 = Dezimalsystem */
  if (strlen(endptr) != 0) {
    /* Da ist noch ein unverarbeiteter Rest */
    printf("Kann '%s' nicht in Zahl umwandeln: Falsches Format\n", text);
  } else if (errno != 0) {
    printf("Kann '%s' nicht in Zahl umwandeln: %s\n", text, strerror(errno));
  } else {
    printf("%ld", zahl);
  }
  return 0;
}
```

Wenn der Text z. B. „779kx“ lautet, dann zeigt `*endptr` nachher auf „kx“ und es wird „Kann '779kx' nicht in Zahl umwandeln: Falsches Format“ gemeldet.
Wenn der Text z. B. „99999999999999999999999999“ lautet, wird die Fehlermeldung „Kann '99999999999999999999999999' nicht in Zahl umwandeln: Numerical result out of range“ ausgegeben.

L.6.4 Weitere Funktionen der Standardbibliothek

Wir werden hier nicht sämtliche Funktionen der Standardbibliothek auflisten, sondern lediglich einen kurzen Überblick geben, damit Sie ein Gefühl für die Möglichkeiten von Standard-C bekommen.

Die Standardbibliothek enthält

- Funktionen zur Ein- und Ausgabe,
- mathematische Funktionen,
- Funktionen zur Verarbeitung von Zeichenketten,
- Funktionen zur Speicherverwaltung,
- etc.

Literaturempfehlungen:

- http://www2.hs-fulda.de/~klingebiel/c-stdlib/index.htm
  - nicht vollständig, aber enthält die wesentlichen Funktionen. Und ist gratis!
  - Als Download auf dem Skriptelaufwerk: fulda-stutz-klingebiel-c-stdlib.tgz
- Online-Buch: Wolf, Jürgen: C von A-Z, 2. Auflage, Galileo 2006, ISBN 3-89842-643-2. Gibt‘s gratis online: http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/
- Sehr gute C-Referenz: Harbison, Samuel P., Steele, Guy L.: C. A Reference Manual, 5th ed., Prentice Hall 2002.
- Klassiker: Kernighan, Brian W., Ritchie, Dennis M.: The C Programming Language, 2nd ed., Prentice Hall 1988. In deutsch zahlreich in der Bibliothek vorhanden.
- Der ANSI-Standard selbst. Eine Vorversion des Standards (Grundlage für ein öffentliches Review) finden Sie auf dem Skriptelaufwerk: ansi_c_draft_19880513.txt.
- Gute Online-Referenz: http://www.cplusplus.com/reference/clibrary/ .

L.6.4.1 Zeichenketten

Headerdatei `ctype.h`

- `isalpha(c)`: prüft, ob das Zeichen c ein Buchstabe ist
- `isupper(c)`: prüft, ob das Zeichen c ein Großbuchstabe ist
- `tolower(c)`: liefert den zugehörigen Kleinbuchstaben
- …

Headerdatei `string.h`

- `strcat(tgt,src)`: hängt Zeichenkette src an tgt an
- `strcmp(s1,s2)`: Vergleicht zwei Strings (Größer? Kleiner? Gleich?)
- `strstr(s1,s2)`: Sucht in s1 nach dem Teilstring s2
- …

L.6.4.2 Mathematische Funktionen

Headerdatei `math.h`

- `pow(x,y)`: berechnet x hoch y
- `sqrt(x)`: berechnet die Quadratwurzel aus x
- `log(x)`: berechnet den natürlichen Logarithmus

- `sin(x)`: Sinus des Bogenmaßes x
- `cos, tan` entsprechend
- `asin, acos, atan`: Zugehörige Umkehrfunktionen
- `abs(x)`: Absolutbetrag von x (ganzzahlig)
- `fabs(x)`: Absolutbetrag des double-Wertes x
- `ceil(x)`: Aufrunden eines double-Wertes
- `floor(x)`: Abrunden
- …

Bei der Verwendung mathematischer Funktionen müssen Sie den Linker mit der zusätzlichen Option `-lm` aufrufen. Dies bindet die mathematischen Funktionen aus der Bibliothek `libm.a` mit ein.

L.6.4.3 Speicherverwaltung

Headerdatei `stdlib.h`

- `malloc(size)`: Speicher anfordern
- `free(p)`: und wieder frei geben
- `calloc(n,size)`: fordert `n` x `size` Bytes an und initialisiert den Bereich mit Nullen
- `realloc(p,size)`: ändert die Größe des dynamisch belegten Speicherbereichs, der die Anfangsadresse `p` hat, zu size Bytes.
- …

Headerdatei string.h

- `memcpy(tgt,src,n)`: kopiert n Bytes von src nach tgt
- `memmove(tgt,src,n)`: Wie `memcpy`. Funktioniert auch bei überlappenden Speicherbereichen.
- `memset(tgt,c,n)`: Speicherbereich `tgt` bis `tgt+n-1` Bytes mit dem Wert `c` überschreiben
- …

L.6.4.4 Uhrzeit

Headerdatei time.h

- `clock()`: liefert einen Zahlenwert für die Prozessorzeit seit dem Start des Programms. Kann genutzt werden, um Laufzeiten zu messen.
- `time(NULL)`: liefert die aktuelle absolute Zeit (Kalenderzeit) als Zahlenwert (normalerweise Anzahl Sekunden seit 1.1.1970, 0 Uhr)
- `localtime(&t)`: erzeugt zu einer absoluten Zeitangabe eine Struktur, die Informationen über diese Zeitangabe enthält. Die zurückgegebene Struktur umfasst die folgenden `int`-Komponenten:
  - `tm_sec` Sekunden seit der letzten vollen Minute.
  - `tm_min` Minuten seit der letzten vollen Stunde.
  - `tm_hour` Stunden seit Mitternacht.
  - `tm_mday` Tage seit Monatsanfang.
  - `tm_mon` Monate seit Jahresbeginn.
  - `tm_year` Jahre seit 1900.
  - `tm_wday` Tage seit dem letzten Sonntag.
  - `tm_yday` Tage seit Jahresbeginn.
- ...

L.6.4.5 Betriebssystemkommandos

Headerdatei stdlib.h

- `system(str)`: Ausführen eines Kommandos auf Betriebssystem-Ebene (zum Beispiel „`ls`“).
- `exit(status)` beendet die Programmausführung und gibt den Ganzzahlwert `status` als Rückkehrstatus an die Aufrufumgebung zurück.

L.6.4.6 Ein-/Ausgabe

Kommt später