6.5 Extras/Funktionszeiger

6.5.1 Situation

Eine Reihe von Messwerten wurde eingelesen. Nun soll von allen diesen Messwerten der Median bestimmt werden. Der Median ist derjenige Messwert, der genau in der Mitte aller Werte liegt: Er ist größer ist als die eine Hälfte und kleiner als die andere Hälfte aller Messwerte.[1]

Wie berechnet man nun den Median? Am einfachsten und schnellsten ist es, die Messwertfolge zu *sortieren* und auf den per Index auf den Wert in der Mitte zuzugreifen.

Nun gibt es verschiedene Sortiermethoden. In der C-Standard-Bibliothek steht eine Funktion bereit, die den so genannten *Quicksort*-Algorithmus benutzt (ein schneller Algorithmus, wenn auch nicht in jedem Fall der schnellste) und die daher `qsort()` genannt wurde.

6.5.2 Die Funktion `qsort` aus der Standard-Bibliothek

Mit `qsort()` kann man ein Array sortieren. Der Prototyp lautet wie folgt:

```
void qsort(void* base, size_t nmemb, size_t size,
           int (*funk)(const void* , const void* );
```

Dabei bezeichnet `base` die Startadresse des Arrays, `nmemb` die Anzahl der Elemente und `size` die Größe eines Elements in Bytes.

`funk` ist die Startadresse einer Vergleichsfunktion, die an `qsort` übergeben wird. `qsort` braucht diese Vergleichsfunktion, um bei Arrays mit komplizierten Elementen (z.B. Strings oder Records) herauszufinden, in welcher Reihenfolge die Elemente sortiert werden sollen. `qsort` selbst ruft die Vergleichsfunktion mehr oder weniger häufig auf (Abbildung 1). Vom Datentyp her ist

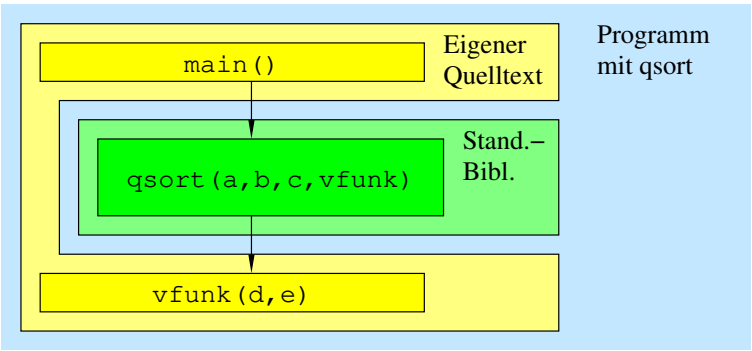

Abbildung 1: Aubau eines Programms mit `qsort`

`funk` ein so genannter *Funktionszeiger*.

6.5.3 Definition eines Funktionszeigers

Ein Zeiger ist immer eine Variable, die für sich keinen Sinn hat, sondern die auf andere Datenelemente (Konstante oder Variable) verweist. In einem Serienbriefprogramm seien z.B. folgende Funktionen vorhanden (siehe `fzbeispiel1.c`):

```
void gruss_nett(void)
{
   printf("Mit freundlichen Gruessen,\n");
}
void gruss_sonst(void)
```

[1] Bei einer geraden Anzahl von Messwerten nimmt man die beiden mittleren Werte und bildet (nur) von diesen beiden den Durchschnitt.

```
{
   printf("Hochachtungsvoll,\n");
}
```

Beide Grußformeln sollen je nach Anwendungsfall eingesetzt werden können. Dann kann man z. B. in `main` eine Variable mit dem Namen `grussfunktion` erstellen:

```
void (*grussfunktion)(void);
```

Die Vereinbarung bedeutet: `grussfunktion` ist der Zeiger auf eine Funktion (Parameterliste: `(void)` mit dem Rückgabetyp `void`. Nun kann man dieser Variablen `grussfunktion` je nach Bedarf eine der oben definierten Funktionen (oder jede andere, die den richtigen Typ hat) zuweisen:

```
grussfunktion = gruss_nett;
```

Eine andere erlaubte Schreibweise mit der gleichen Bedeutung ist:

```
grussfunktion = &gruss_nett;
```

Wichtig ist hier, dass weder bei `grussfunktion` noch bei `gruss_nett` ein Klammernpaar steht:

```
gruss_nett(); /* bewirkt den Aufruf der genannten Funktion. */
gruss_nett; /* gibt die Startadresse der Funktion zurueck. */
```

Mit Hilfe des Funktionszeigers kann man nun die gemeinte Funktion aufrufen:

```
grussfunktion();
```

In alten Quelltexten findet man dafür die Schreibweise (mit der gleichen Bedeutung):

```
(*grussfunktion)();
```

Insgesamt sieht der Quelltext also so aus:

```
#include <stdio.h>

void gruss_nett(void)
{
    printf("Mit freundlichen Gruessen,\n");
}

void gruss_sonst(void)
{
    printf("Hochachtungsvoll,\n");
}

int main(void)
{
    void (*grussfunktion)(void);
    grussfunktion = gruss_nett;

    printf("Sehr geehrter Herr Meier,\n");
    printf("Leider kann ich an Ihrem Seminar mit dem Titel\n ");
    printf("'Abmahnen fuer Anfaenger' nicht teilnehmen.\n");
    grussfunktion();
    printf("P. Mueller\n");
    return 0;
}
```

6.5.4 Von der Funktion zum Funktionszeiger

Wie kann man nun aus dem Prototypen einer Funktion einen Funktionszeiger basteln? Bei normalen Zeigern war das einfach: Wenn `double x;` die Variable ist, auf die ich zeigen will, dann ist `double* px;` der entsprechende Zeiger.

Bei Funktionszeigern geht man so vor – am Beispiel der Sinusfunktion aus `math.h` vorgestellt:

```
double sin(double x);  /* Das ist der Prototyp */
double sin(double);    /* Alle Parameternamen weggelassen */
double *sin(double);   /* Stern um den Funktionsnamen */
double (*sin)(double); /* Klammern um Stern und Funktionsnamen */
```

Warum reicht nicht einfach der Stern, wozu sind die Klammern dar? Die vorletzte Zeile deklariert eine Funktion, die als Rückgabewert Zeiger auf `double` hat, ist also ein Prototyp — der Stern ist also stärker an `double` als an `sin` gebunden. Die letzte Zeile dagegen ist tatsächlich der *Zeiger auf eine Funktion* mit dem Rückgabewert `double`.

6.5.5 Funktionszeiger in der Parameterliste

Ähnlich wie normale Zeiger haben auch Funktionszeiger an sich keinen besonderen Nutzen. Interessant werden sie erst, wenn sie per Parameter an Funktionen übergeben werden.

Ein einfaches Beispiel zeigt folgender Quelltext `fzbeispiel2.c`:

```c
#include <stdio.h>
#include <math.h>

double funkaufruf(double (*funk)(double), double zahl)
{
    double x;
    x=funk(zahl);
    return x;
}

int main(void)
{
    double a;
    a=funkaufruf(sin, 3.14159);
    printf("Ergebnis:%lf\n", a);
    return 0;
}
```

Die Funktion `funkaufruf()` nimmt als ersten Parameter einen Funktionszeiger (z.B. einen Zeiger auf die Sinus-Funktion `sin`) und als zweiten Parameter eine Zahl (z.B. π). Sie ruft intern die Funktion mit der Zahl auf (also $\sin \pi$) und gibt das Ergebnis wiederum an den Aufrufer zurück.

Auf diese Art kann man einer Funktion nicht nur simple Daten mitgeben (im Beispiel eine Zahl), sondern auch einen beliebigen, evtl. selbst geschriebenen Auftrag (im Beispiel die Sinus-Funktion). Dieser Auftrag muss nicht bekannt sein, wenn man die Funktion `funkaufruf()` schreibt.

Ein Beispiel aus der C-Standardbibliothek ist die Funktion `atexit()`. Mit dieser Funktion kann man bestimmen, welche Befehle bei einem normalen Programmende (z.B. durch `return` aus `main()` oder durch Aufruf von `exit()`) ausgeführt werden sollen. Diese Befehle schreibt man in eine Funktion `ende()` hinein (siehe `atexit_beispiel.c`):

```c
#include <stdio.h>
#include <stdlib.h>
void ende(void)
{
```

```
5     printf("Das war's dann.\n");
6     sleep(1);
7     printf("Tschuess!\n");
8  }
9  int main(void)
10 {
11    atexit(ende);
12    printf("Mein Programm laeuft.\n");
13    sleep(1);
14    printf("Es laeuft immer noch.\n");
15    sleep(1);
16    return 0;
17 }
```

Die Funktion `ende` wird nun durch einen Aufruf von `atexit()` registriert (angemeldet), indem man die Funktion `ende` als Parameter an `atexit()` übergibt. Sobald das Programm nun beendet wird, wird die Funktion `ende` aufgerufen. Man kann übrigens mehrere solcher Funktionen (bis zu 32) in einem Programm registrieren.

Schwierig an `atexit()` ist nur die Deklaration:

```
int atexit(void (*function)(void));
```

`atexit()` ist eine Funktion, die als Parameter einen Funktionszeiger hat, der auf eine Funktion zeigt, die die Parameterliste `(void)` und den Rückgabetyp `void` hat.

6.5.6 Vergleichsfunktion für `qsort()`

`qsort()` hat ebenfalls einen Funktionszeiger als Parameter:

```
int (*funk)(const void* , const void*);
```

`funk` ist also eine Funktion `(const void*, const void*)` mit Rückgabetyp `int`. Für die beiden zu vergleichenden Elemente des Arrays werden also zwei Zeiger übergeben – falls die Elemente sehr groß sind, ist das effizienter, als die Inhalte selbst zu übergeben. Eine Vergleichsfunktion `intvergl()` hat also den Prototyp:

```
int intvergl(const void* pa, const void* pb);
```

Die Definition von `intvergl()` ist dann:

```
1  int intvergl(const void *pa, const void *pb)
2  {
3     int* pi = (int*) pa;
4     int* pj = (int*) pb;
5     if(*pi > *pj) return 1;
6     if(*pi < *pj) return -1;
7     return 0;
8  }
```

Zuerst werden die allgemeinen Zeiger `pa` und `pb` in `int`-Zeiger `pa` und `pb`umgewandelt; anschließend werden die Ziele verglichen. Der Aufruf von `qsort()` ist dann problemlos:

```
1  int main(void)
2  {
3     int werte[4]={20,31,10,50};
4     qsort(werte, 4, sizeof(werte[0]), intvergl);
5     /* ... */
6  }
```

Der gesamte Quelltext befindet sich in `qsortbeispiel1.c`.

6.5.7 Wo braucht man Funktionszeiger?

Funktionszeiger sind immer dann sinnvoll, wenn eine Bibliotheksfunktion nicht wissen kann, mit welchen anderen Funktionen sie in der Zukunft zusammenarbeiten muss. Dann hilft auch keine Fallauswahl (`switch - case`): Man könnte `qsort()` durchaus beibringen, int, float, double usw. zu sortieren, evtl. auch Zeichenketten. Spätestens bei Records ist aber Schluss: Es ist unklar, nach welchen Komponenten irgendein Record sortiert werden soll. Hier sind Funktionszeiger eine echte Hilfe.

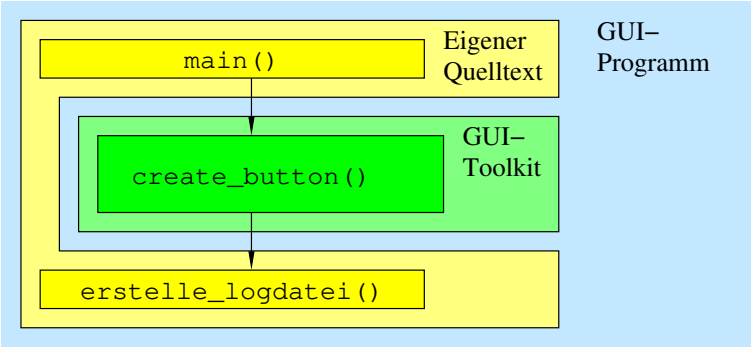

Abbildung 2: Aubau eines Programms mit GUI

Eine andere Anwendung findet statt bei Betriebssystemen und graphischen Benutzeroberflächen. Dort erwarten viele Funktionen die Angabe einer Callback-Funktion: Man teilt als Anwendungsprogrammierer dem System mit, welche Funktion bei einem Mausklick auf einem bestimmten Feld aufgerufen werden soll (indem man die Adresse der Funktion angibt). Wenn dann zur Laufzeit der Mausklick stattfindet, wird die Funktion mit dieser Adresse aufgerufen – das System führt dann meine Funktion aus (Abbildung 2).