Übung Nr. 3:

Es soll das Programm „BitTest“ erstellt werden, mit dem nach Eingabe einer ganzen oder einer gebrochenen Dezimalzahl ihre rechnerinterne Binärdarstellung ausgegeben wird (Abb. 1). Ein Grundgerüst dafür kann unter http://homepages.thm.de/christ/ heruntergeladen werden.

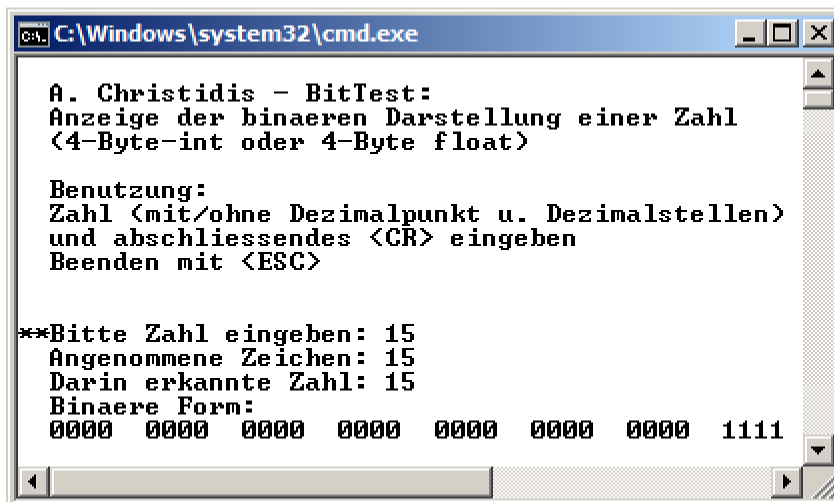

Abb. 1 BitTest.exe (Start)

Damit erst während der Eingabe entschieden werden kann, ob eine ganze oder eine gebrochene Zahl gewünscht ist, sollen die eingetippten Schriftzeichen ausgewertet werden. Dabei soll durch einzelne Autokorrektur-Maßnahmen auch das Komma als Dezimalzeichen zugelassen werden; umgekehrt sollen einzelne unpassende Eingaben zu Fehlermeldungen führen (Abb. 2).

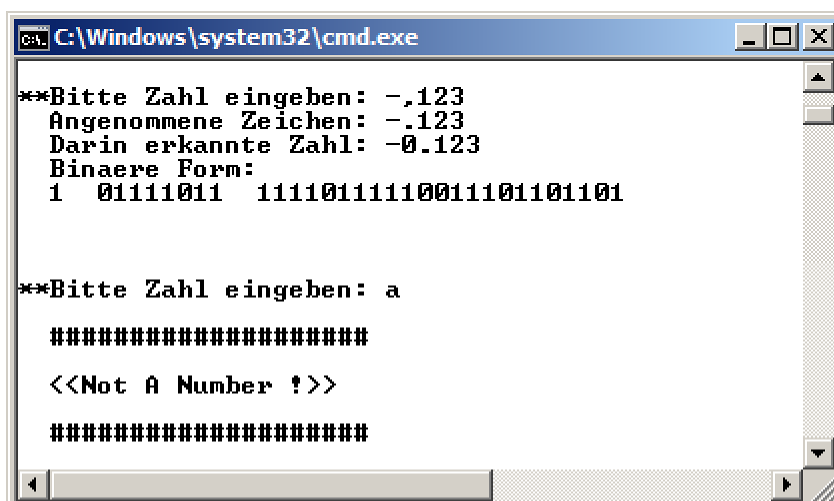

Abb. 2 BitTest.exe (Autokorrektur)

Eigenentwicklungen sind willkommen; es wird jedoch empfohlen, den Ausbau von BitTest bis zur geforderten Funktionalität in der Reihenfolge der im folgenden wiedergegebenen Schritte vorzunehmen. Zur Erleichterung der Arbeit sind die Stellen, die zur Veränderung des Codes geeignet sind, mit numerierten Direktiven-Paaren (`#ifdef`, `#endif`) versehen.

Zur Struktur von BitTest

Das Programm besteht aus fünf Funktionen:

`main()` dient der Koordination des ganzen Geschehens: Es eröffnet den Programmlauf und ruft in einer Endlosschleife nacheinander die Funktionen zur Eingabe einer ganzzahligen oder gebrochenen Dezimalzahl und zur Ermittlung ihrer Bitbelegung auf. Die Funktion ist vollständig; gleichwohl verändert sie sich im Laufe der Programmentwicklung durch die vorgesehene bedingte Compilierung.

`errMsg()` ist „gebrauchsfertig"; sie enthält lediglich den Wortlaut und die Formatierung von Fehlermeldungen.

`gebein()` soll die eingetippten Zeichen bis zum `<CR>` (Carriage Return, Enter-Taste) lesen und daraus eine Zahl bilden, die sie über eine der beiden übergebenen Adreß-Dummy-variablen `fdum` oder `idum` zurückgibt, je nachdem, ob ein gebrochener Teil (getrennt durch einen Dezimalpunkt) eingegeben wurde oder nicht. Die Funktion, von der `gebein()` aufgerufen wurde, erfährt über den Rückgabewert, von welchem Datentyp die Eingabe war (und somit welche der beiden übergebenen Variablen das Ergebnis enthält). `gebein()` hat bereits einen Großteil ihrer Funktionalität; sie muß aber noch ausgebaut werden (ca. 15 Quellcode-Zeilen).

`binFormU()` soll, abhängig vom Wert der Variablen `type`, der `FLT` (für `float`) oder `INT` (für `int`) betragen kann, die interne binäre Darstellung einer `float`- oder einer `int`-Variablen wiedergeben. Sie tut dies mit Hilfe der Union `floint`, die bewirkt, daß eine `int`- und eine `float`-Variable denselben physikalischen Speicherplatz teilen. An dieser Funktion muß noch gearbeitet werden (ca. 2 Zeilen).

`binFormP()` schließlich soll exakt die gleiche Wirkung wie `binFormU()` entfalten. Statt mit einer Union soll sie jedoch diese Aufgabe lediglich mit einer `int`-Adreßvariablen bewerkstelligen. Diese Funktion muß noch größtenteils programmiert werden (max. 10 Zeilen).

Die hier beschriebene Bearbeitung kann beschritten werden, indem der Quellcode vervollständigt wird. Teile der Lösung sind vorhanden; sie können aktiviert werden, indem die Auskommentierung der Bezeichner `MORE_BIT1` bis `MORE_BIT6` in der Header-Datei (am einfachsten: nacheinander) aufgehoben wird. Im anfänglichen (gelieferten) Zustand sind diese Bezeichner unwirksam (auskommentiert).

1. Compilierbarkeit des Programms

Das Programm ist im gelieferten Zustand unvollständig, aber korrekt codiert; dennoch läßt es sich nicht compilieren: Der Compiler meldet einen Konflikt um die Funktion `errMsg()`. Der Grund dafür ist, daß, bevor der sequentielle Compilierungsprozeß den Funktionscode von `errMsg()` erreicht, schon in `gebein()` seinen Aufruf antraf. Aufgrund seiner Voreinstellung geht dann der Compiler davon aus, daß er im weiteren Procedere eine Funktion mit dem Namen `errMsg()` und einem (standardmäßig angenommenen) `int`-Rückgabewert vorfindet. Bei der Feststellung, daß `errMsg()` vom Typ `void` und somit ohne Rückgabewert ist, meldet er einen Fehler. Dieser wird schnell behoben, indem die Prototypen der Funktionen angemeldet werden, vorzugsweise in der Header-Datei (`BitTest.h`).

Eine geeignete Position ist dort mit `MORE_BIT1` markiert.[1]

2. Vervollständigung von `gebein()`

Nach der ersten Compilierung zeigt `gebein()` schon einen Großteil der zugedachten Funktionalität: Zeichen (erwartete Ziffern einer Zahl) werden über die Tastatur in einer endlosen `while`-Schleife in eine indizierte Puffer-Variable (`buf[]`) eingelesen. [Hier lohnt es sich, nachzudenken, warum es nicht ratsam ist, eine `do-while`-Schleife zu benutzen.]

Nach Verlassen der `while`-Schleife mit `<CR>` wird der Puffer einer der beiden Funktionen `atoi()` bzw. `atof()` übergeben. Beide gehören zum Sprachumfang von C und wandeln ASCII-Zeichen eines Strings jeweils in eine `int`- bzw. eine `float`-Zahl um. Die Entscheidung darüber, welche der beiden Funktionen zu verwenden ist, wird über die Erkennung eines Dezimalpunktes getroffen, nachdem anfänglich vom Typ `int` ausgegangen wird (Variable `flint` mit Werten aus den `enum`-Bezeichnern in der Header-Datei).

Die Funktionalität und der Programmierstil von `gebein()` sind zu verbessern:

Obwohl die eingegebene Zahl richtig erkannt und wiedergegeben wird, erscheinen Sonderzeichen bei der Wiedergabe der eingelesenen Ziffern (Abb. 3). Das liegt daran, daß der Puffer keine abschließende Escape-Sequenz (`'\0'`) enthält. Ein passender Platz dafür ist mit dem Bezeichner `MORE_BIT2` markiert.

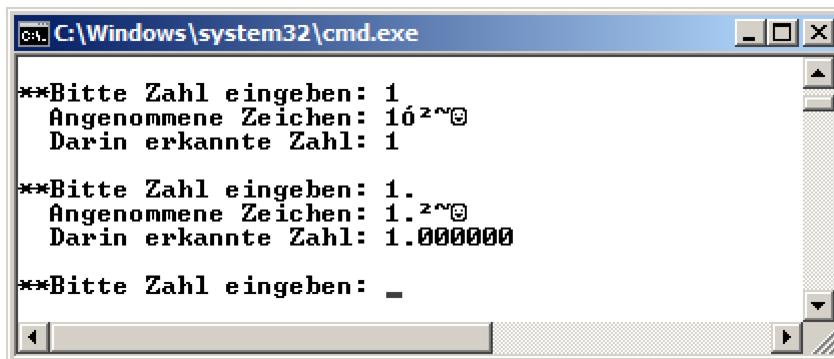

Abb. 3 BitTest.exe ohne Escape-Sequenz

Als nächstes sollte dafür gesorgt werden,

- daß auch das Dezimalkomma als Dezimalpunkt akzeptiert wird,
- daß eine wiederholte Eingabe von Dezimalzeichen zur Zurückweisung als nicht-numerische Eingabe führt (Fehlermeldung `NAN`),
- daß nach Betätigung der Rücktaste die Länge der eingegebenen Zeichenkette nicht kleiner als Null werden kann,
- daß mit der Rücktaste nicht nur der Cursor rückpositioniert, sondern auch das darüber liegende Zeichen gelöscht wird, und

[1] Unter einem Prototyp ist die Deklarationszeile (Name und Parameterliste) einer Funktion gemeint, gefolgt von einem Semikolon. Er teilt dem Compiler mit, welche Funktionen im gesamten Programm enthalten sind, um eben solchen Situationen wie der vorliegenden vorzubeugen.

- daß durch flexibles Format in `printf()` eine erkannte `float`-Zahl mit mindestens einer Vor- und einer Nachkommastelle ausgegeben wird, sonst aber nicht länger, als sie eingegeben wurde.

Dafür geeignete Stellen im Programm sind an Direktiven mit `MORE_BIT3` zu erkennen.

Das Programmverhalten nach diesem Schritt ist in Abb. 4 dokumentiert.

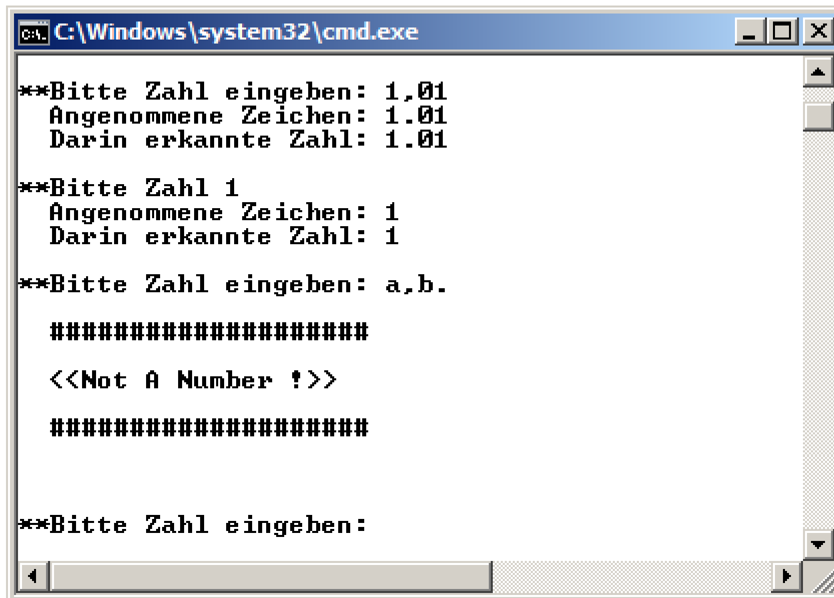

Abb. 4 BitTest.exe nach weiterer Korrektur

Die in Abb.4 wiedergegebene Zurückweisung der Zeichenfolge beruhte auf der Eingabe von zwei Dezimalzeichen. Eine Zurückweisung nicht-numerischer Zeichen erfolgt nur bei Eingabe eines Leerzeichens. Das bestimmt die Direktive `NOTANUMBER` in der Header-Datei (gekennzeichnet mit `MORE_BIT4`). Sie ist auszudehnen auf alle Zeichen, die nicht zwischen 0 und 9 liegen, abgesehen vom Dezimalzeichen und den beiden Vorzeichen.

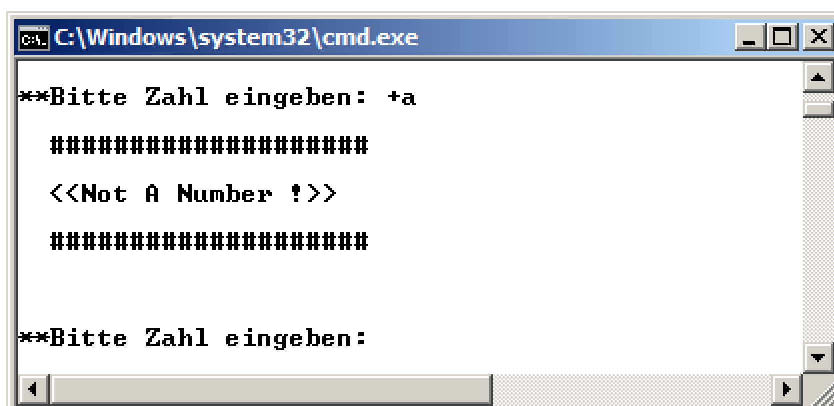

Abb. 5 Zurückweisung nicht-numerischer Zeichen

Die verlangte Funktionalität von `gebein()` ist damit erreicht. Die vielen `if`-Abfragen sollen noch übersichtlich zu einer `switch`-Anweisung zusammenzufaßt werden. Ein passender Platz ist vorgesehen und mit `MORE_BIT4` gekennzeichnet. Dann kann durch bedingte Compilierung zwischen beiden Codierungsarten gewechselt und verglichen werden.

Hinweise:

a) Manchmal gelingt es, eine `switch`-Anweisung so zu gestalten, daß Fallunterscheidungen mit übereinstimmenden Maßnahmen (z.B.: gleiche Wirkung beim Tippen großer und kleiner Buchstaben) untereinander gelistet und mit einer gemeinsamen `break`-Anweisung beendet werden.

b) Bei Auslassung der `break`-Anweisung im unteren Teil der Liste führen die zuletzt abgehandelten Fälle zu einem gemeinsamen `default`-Block.

Die so implementierte Funktion braucht nicht weiter ausgebaut zu werden. Dennoch registriert sie nicht, ob von der Betätigung der Löschtaste der Dezimalpunkt betroffen war; deshalb führt die einmal gedrückte Punkt- oder Kommataste dazu, daß die schließlich gebildete Zahl als `float` interpretiert wird. Ebenso werden nicht-darstellbar große `int`- oder zu kleine `float`-Werte nicht „abgefangen“ (d.h.: ihre Eingabe wird nicht verhindert). Es wird lediglich die Einhaltung der Eingabe von maximal `NCHAR` Zeichen überprüft (Abb. 6).

```
C:\Windows\system32\cmd.exe

**Bitte Zahl eingeben: 12345
  Angenommene Zeichen: 12345
  Darin erkannte Zahl: 12345.0

**Bitte Zahl eingeben: 999999999999
  Angenommene Zeichen: 999999999999
  Darin erkannte Zahl: 2147483647

**Bitte Zahl eingeben: 999999999999999999999

  #####################

  <<Number too long !>>

  #####################


**Bitte Zahl eingeben:
```

Abb. 6 Zur Funktionalität von `gebein()`

(Frage: Welchen Bezug hat die Zahl in Abb. 6 zur Berechnung 2^{32} = 4.294.967.296 der Kombinationsmöglichkeiten von 32 Bit?)

3. Vervollständigung von `binFormU()`

Aktivierung des Bezeichners `MORE_BIT5` bewirkt in `main()`, daß (abhängig vom Rückgabewert von `gebein()`) `binFormU()` aufgerufen wird. Die Funktion soll (abhängig vom Wert in der Variablen `type`) die Bitbelegung einer von zwei übergebenen Variablen in einer Folge von „0“ und „1“ darstellen.

Wichtig in diesem Zusammenhang sind die sog. „Shift-Operatoren“ „>>“ und „<<“; sie verschieben die Bits des linken Operators um so viele Stellen, wie der rechte Operator angibt (die Operatoren sind jeweils Zahlen oder Variablen); `x<<y` bedeutet somit für zwei ganzzahlige Variablen `x` und `y`, daß `x` mit 2^y multipliziert wird. Die umgekehrte Wirkung (Division) hat der Operator `>>`. Ein Anwendungsbeispiel ist in der provisorischen Version realisiert.

Um ein Bit an beliebiger Stelle einer Zahl oder Variablen auszulesen, braucht man die bitweisen Vergleichsoperatoren; die wichtigsten davon sind das bitweise UND („&“) und das bitweise ODER („|“):

Das Ergebnis der Operation `(x&y)` ist ≠ 0 (d.h. in C auch: wahr), wenn der Binär-Wert von `x` und jener von `y` an mindestens einer Stelle gemeinsam ein gesetztes Bit (1) haben; so extrahiert z.B. der Ausdruck `(x&1)` das niedrigstwertige Bit der Variablen `x`, d.h. die Information, ob `x` ungerade ist.

Das Ergebnis der Operation `(x|y)` ist 1 (d.h. in C: wahr), wenn mindestens einer der Werte `x` oder `y` ein gesetztes Bit (1) hat, d.h., wenn mindestens einer der beiden ≠ 0 ist; es ist =0 nur, wenn beide Größen null sind.

Da die Operatoren der bitweisen Verknüpfungen ganzzahlig sein müssen, `binFormU()` aber auch die Bitbesetzung von `float`-Zahlen wiedergeben soll, hält `binFormU()` die Union einer `float`- und einer `int`-Variablen bereit, damit sie anhand der letzteren die Bitbelegung des (gemeinsam genutzten) Speicherplatzes ermitteln kann.

Auf dieser Grundlage ist die Funktion `binFormU()` zu vervollständigen. Im Ergebnis liefert BitTest Ergebnisse wie in Abb. 7.

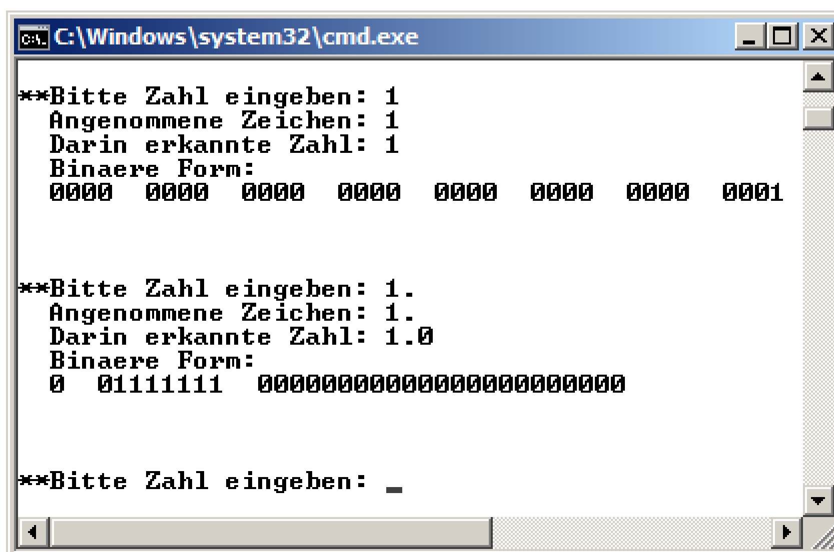

Abb. 7 BitTest.exe bei voller Funktionalität

4. Vervollständigung von `binFormP()`

Die Eleganz der Lösung mit einer Union darf nicht darüber hinwegtäuschen, daß für diese einfache Aufgabe auch schon eine `int`-Adreßvariable gereicht hätte. Diese ist in `binFormP()` bereits eingerichtet. Der Code kann dann frei entwickelt werden. Durch Aktivierung des Bezeichners `MORE_BIT6` ist auch in `main()` dafür gesorgt, daß diese Funktion zu passenden Werten von `gebein()` aufgerufen wird. Dabei ist es für die Funktionalität gleichgültig, ob nur eines von `MORE_BIT5` und `MORE_BIT6` oder beide aktiv sind (etwa zum Vergleich der Ergebnisse, die dann doppelt erscheinen).

Die Aufgaben sind zu lösen und in der o.a. Reihenfolge (z.B. als Reproduktin der Abbildungen) vorzustellen. Wichtig dabei: die Inanspruchnahme des Autors, falls sich Ratlosigkeit einstellen sollte. ☺