SWO - WS 14/15 - Übung 1

Christian Kapplmüller - S1317307017

25. September 2014

Inhaltsverzeichnis

1 1.1 Dreieck-Tester **2**
1.1 Lösungsansatz . . . 2
1.1.1 Vergleiche . . . 2
1.2 Sourcecode . . . 2
1.3 Tests . . . 4
1.3.1 Ungültige Parameterzahl . . . 4
1.3.2 Ungültiger Wertebereich . . . 4
1.3.3 Ungültiges Dreieck . . . 5
1.3.4 Rechtwinkelig Dreieck . . . 5
1.3.5 Gleichschenkeliges Dreieck . . . 6
1.3.6 Gleichseitiges Dreieck . . . 6
1.3.7 Rechtwinkelig-gleichschenkeliges Dreieck . . . 7

2 1.2 Primfaktorenzerlegung **8**
2.1 Lösungsansatz . . . 8
2.2 Sourcecode . . . 8
2.3 Tests . . . 10
2.3.1 Ungültige Parameterzahl . . . 10
2.3.2 Nichtnumerischer Parameter . . . 10
2.3.3 Ungültiger Wertebereich . . . 10
2.3.4 Einzelner Primfaktor . . . 11
2.4 Mehrere Primfaktoren . . . 11

3 1.2 ERRNO & Co **12**
3.1 Fehlende fehlerbehandlung von atoi . . . 12
3.2 Erörterungen zu ERRNO . . . 12

1 1.1 Dreieck-Tester

1.1 Lösungsansatz

Als Eingabeüberprüfung werden Anzahl der Parameter und ihr Wertebereich geprüft. Das Programm unterstützt Fließkommawerte („double“). Um mit Fließkommawerten exakt rechnen (und die Werte vergleichen) zu können, ist eine Konstante EPSILON notwendig, die definiert, mit welcher Genauigkeit gearbeitet werden soll. Daher werden Zahlenvergleiche (z.B. a == b) als Vergleich mit EPSILON ausgeführt (a - b < EPSILON).

1.1.1 Vergleiche

Gültiges Dreieck	Für ein gültiges Dreieck darf die Summe zweier beliebiger Seiten die dritte Seine nicht übersteigen
Quadratisches Dreieck	Die Prüfung auf ein rechtwinkeliges Dreieck erfolgt durch durchprobieren aller möglichen Kombinationsmöglichkeiten und Einsetzten im Satz des Pythagoras. Wenn bei einer Kombination die Gleichung wahr ist, so ist das Dreieck quadratisch.
Gleichschenkeliges Dreieck	Für ein gleichseitiges Dreieck müssen mindestens 2 der Eingabewerte gleich sein.
Gleichseitiges Dreieck	Hier müssen alle 3 Eingabewerte gleich sein.

1.2 Sourcecode

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* using a small calculation for EPSILON to make its
5      smallness more visible, used for double-compares */
6  #define EPSILON 1.0 / 1000.0 / 1000.0 / 1000.0 / 1000.0
7
8  typedef enum {false, true} bool;
9
10 double computeSquare(double value) {
11   return value * value;
12 }
13 double absDouble(double value) {
14   /* using a "inverted" style of comparision
15     to avoid "if (a = 2)" bugs */
16   return 0.0 <= value ? value : value * -1.0;
17 }
18 void printUsage(char programName[]) {
19   printf("USAGE: %s sideLenghOfTriange sideLenghOfTriange sideLenghOfTriange\n",
20             programName);
21 }
22
23 int main(int argc, char *argv[]) {
24   double triangle[3];
```

```
25  int i;
26  bool valid;         /* gueltig */
27  bool square;        /* quadratisch */
28  bool isosceles;     /* gleichschenkelig */
29  bool equilateral;   /* gleichseitig */
30
31  if (4 != argc) {
32    printf("provide exactly 3 parameters (%d provided)\n", argc - 1);
33    printUsage(argv[0]);
34    return EXIT_FAILURE;
35  }
36
37  for (i = 0; i < 3; i++) {
38    sscanf(argv[i + 1], "%lf", &triangle[i]);
39    if (0 >= triangle[i]) {
40      printf("argument number %d was 0 or negative\n", i + 1);
41      printUsage(argv[0]);
42      return EXIT_FAILURE;
43    }
44  }
45  valid = true;
46  for (i = 0; i < 3; i++) {
47    /* using the calculations 3 % 3 = 0 and 4 % 3 = 1 to "roundtrip" the array and
48      validate every value of the given triangle */
49    if (triangle[i] + triangle[(i + 1) % 3] <= triangle[(i + 2) % 3] + EPSILON) {
50      valid = false;
51    }
52  }
53  /* a invalid triangle has non of the following three attributes */
54  equilateral = false;
55  isosceles = false;
56  square = false;
57  if (valid) {
58    equilateral = true;
59    for (i = 0; i < 3; i++) {
60      if (absDouble(computeSquare(triangle[i]) + computeSquare(triangle[(i + 1) % 3])
61                - computeSquare(triangle[(i + 2) % 3])) < EPSILON) {
62        square = true;
63      }
64      if (absDouble(triangle[i] - triangle[(i + 1) % 3]) < EPSILON) {
65        isosceles = true;
66      }
67      if (absDouble(triangle[0] - triangle[i]) > EPSILON) {
68        equilateral = false;
69      }
70    }
71  }
72  printf("valid: %s\n", valid ? "Yes" : "No");
73  printf("equilateral: %s\n", equilateral ? "Yes" : "No");
74  printf("isosceles: %s\n", isosceles ? "Yes" : "No");
75  printf("square: %s\n", square ? "Yes" : "No");
76  return EXIT_SUCCESS;
77 }
```

1.3 Tests

1.3.1 Ungültige Parameterzahl

Eingabewerte:

- 1 2 3 4
- [keine Parameter]

Ausgabe:

```
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 1 2 3 4
provide exactly 3 parameters (4 provided)
USAGE: ./uebung1_1 sideLenghOfTriange sideLenghOfTriange sideLenghOfTriange
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1
provide exactly 3 parameters (0 provided)
USAGE: ./uebung1_1 sideLenghOfTriange sideLenghOfTriange sideLenghOfTriange
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$
```

Abbildung 1: Ungültige Parameterzahl (Dreieckszerlegung)

Bei allen Testfällen wird hier die Fehlermeldung für eine ungültige Parameteranzahl incl. usage ausgegeben.

1.3.2 Ungültiger Wertebereich

Eingabewerte:

- 0 0 0
- -1 2 -3
- 2 -2 2

Ausgabe:

```
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 0 0 0
argument number 1 was 0 or negative
USAGE: ./uebung1_1 sideLenghOfTriange sideLenghOfTriange sideLenghOfTriange
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 -1 2 -3
argument number 1 was 0 or negative
USAGE: ./uebung1_1 sideLenghOfTriange sideLenghOfTriange sideLenghOfTriange
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 2 -2 2
argument number 2 was 0 or negative
USAGE: ./uebung1_1 sideLenghOfTriange sideLenghOfTriange sideLenghOfTriange
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$
```

Abbildung 2: Ungültiger Wertebereich (Dreieckszerlegung)

Wie erwartet wird hier die Fehlermeldung für den ersten invaliden Wert die Fehlermeldung für einen ungültigen Wertebereich incl. usage ausgegeben.

1.3.3 Ungültiges Dreieck

Eingabewerte:

- 10 10 100
- 0,1 0,7 0,3
- 11,1 1,1 9,9

Ausgabe:

```
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 10 10 100
valid: No
equilateral: No
isosceles: No
square: No
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 0.1 0.7 0.3
valid: No
equilateral: No
isosceles: No
square: No
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 11.1 1.1 9,9
valid: No
equilateral: No
isosceles: No
square: No
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$
```

Abbildung 3: Ungültiges Dreieck

Bei allen Testfällen wird „valid: No“ ausgegeben.

1.3.4 Rechtwinkelig Dreieck

Eingabewerte:

- 3 4 5
- 0,04 0,03 0,05
- 2,5 1,5 2

Ausgabe:

```
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 3 4 5
valid: Yes
equilateral: No
isosceles: No
square: Yes
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 0.04 0.03 0.05
valid: Yes
equilateral: No
isosceles: No
square: Yes
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 2.5 1.5 2
valid: Yes
equilateral: No
isosceles: No
square: Yes
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$
```

Abbildung 4: Rechtwinkeliges Dreieck

Bei allen Testfällen wird „valid: Yes“ und „square: Yes“ ausgegeben.

1.3.5 Gleichschenkeliges Dreieck

Eingabewerte:

- 35.8495 50.07 35.8495
- 50 50 1
- 17 14.5268 14.5268

Ausgabe:

```
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 35.8495 50.07 35.8495
valid: Yes
equilateral: No
isosceles: Yes
square: No
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 50 50 1
valid: Yes
equilateral: No
isosceles: Yes
square: No
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_1 17 14.5268 14.5268
valid: Yes
equilateral: No
isosceles: Yes
square: No
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$
```

Abbildung 5: Gleichschenkeliges Dreieck

Bei allen Testfällen wird „valid: Yes“ und „isosceles: Yes“ ausgegeben.

1.3.6 Gleichseitiges Dreieck

Eingabewerte:

- 123.456789 123.456789 123.456789

- 1 1 1
- 0.00012 0.00012 0.00012

Ausgabe:

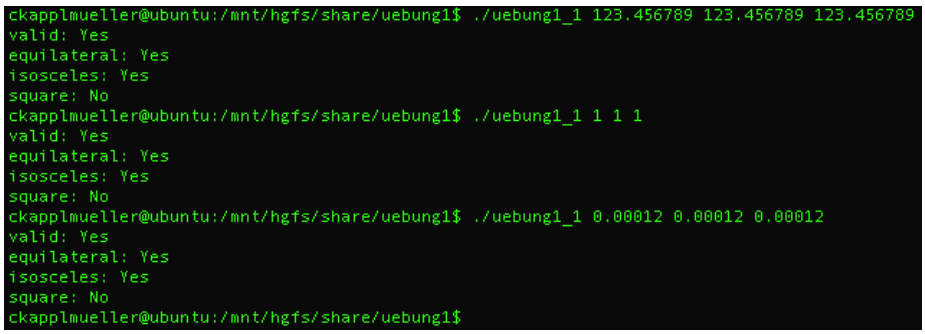

Abbildung 6: Gleichseitiges Dreieck

Bei allen Testfällen wird „valid: Yes“ und „equilateral: Yes“ ausgegeben. Da es in der Natur der Sache liegt, dass gleichseitige Dreiecke auch gleichschenkelig ist, wird zusätzlich auch „isocseles: Yes“ ausgegeben.

1.3.7 Rechtwinkelig-gleichschenkeliges Dreieck

Eingabewerte:

- Hinweis: $\sqrt{2} \approx 1.41421356237309504880168872420969807856967187537694 80$
  Nachstehend mit „1.4142...“ bezeichnet.
- 1 1 1.4142...
- 1.4142... 1.4142... 2
- „1“41.42... 200 141.42...

Ausgabe:

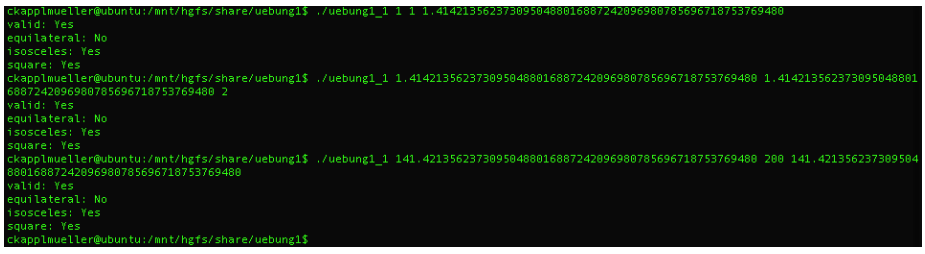

Abbildung 7: Rechtwinkelig-gleichschenkeliges Dreieck

Bei allen Testfällen wird „valid: Yes“, „isosceles: Yes“ und „square: Yes“ ausgegeben.

2 1.2 Primfaktorenzerlegung

2.1 Lösungsansatz

Als Algorithmus für diese Aufgabe wurde die Primfaktorenzerlegung via Probedivision gewählt.
Zur Parameterprüfung sind folgende Szenarien zu berücksichtigen:

- Ungültige Parameterzahl (keiner oder mehr als einer)
- Nicht parsebare Zeichen im Eingabewert
- Nicht parsebarer Eingabewert auf Grund von Datentypbeschränkungen (int)
- Eingabewert ist kleinergleich 0
- Eingabewert ist ungerade

2 ist die kleinste Primzahl, und damit der Startwert für die Basis der Exponentenschreibweise.

Ist der Eingabewert durch die Basis ohne Rest teilbar, so wird der Exponent solange inkrementiert und der Eingabewert durch die Basis dividiert, bis die Teilbarkeit durch die Basis nicht mehr gegeben ist. In diesem Fall wird der erste Primfaktor in Exponentenschreibweise ausgegeben. Nun wird die Basis inkrementiert und der Exponent zurückgesetzt.

Dies wird solange wiederholt, bis der Eingabewert bei 1 angelangt ist. Zu guter letzt wird der letzte (bzw. einzige) Primfator in Exponentenschreibweise ausgegeben.

2.2 Sourcecode

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

typedef enum {false, true} bool;

void printUsage(char programName[]) {
  printf("USAGE: %s prositiveEvenValue\n", programName);
}

int main(int argc, char *argv[]) {
  int base;
  int exponent;
  int input;
  char *garbage = "\0";

  /* using a "inverted" style of comparision
    to avoid "if (a = 2)" bugs */
```

```
20   if (2 != argc) {
21     printf("invalid number of parameters\n");
22     printUsage(argv[0]);
23     return EXIT_FAILURE;
24   }
25
26   errno = 0;
27   input = strtol(argv[1], &garbage, 10);
28   /* using a separate comparision although "if (errno)"
29     would do the job to make the source better readable */
30     if (0 != errno) {
31       printf("parsing of input value failed: %s\n", strerror(errno));
32       printUsage(argv[0]);
33       return EXIT_FAILURE;
34     }
35     if (0 < strlen(garbage)) {
36       printf("unparsable part of input value: %s\n", garbage);
37       printUsage(argv[0]);
38       return EXIT_FAILURE;
39     }
40
41   if (0 >= input) {
42     printf("input value zero or negative\n");
43     printUsage(argv[0]);
44     return EXIT_FAILURE;
45   }
46
47   if (0 != input % 2) {
48     printf("input value is odd\n");
49     printUsage(argv[0]);
50     return EXIT_FAILURE;
51   }
52
53   base = 2;
54   exponent = 0;
55   while (1 != input) {
56     if (0 == input % base) {
57       exponent++;
58       input /= base;
59     } else {
60       if (0 != exponent) {
61         printf("%d^%d * ", base, exponent);
62       }
63       base++;
64       exponent = 0;
65     }
66   }
67   printf("%d^%d\n", base, exponent);
68
69   return EXIT_SUCCESS;
70 }
```

2.3 Tests

2.3.1 Ungültige Parameterzahl

Eingabewerte:

- 2 4
- [keine Parameter]

Ausgabe:

```
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_2 2 4
invalid number of parameters
USAGE: ./uebung1_2 prositiveEvenValue
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_2
invalid number of parameters
USAGE: ./uebung1_2 prositiveEvenValue
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$
```

Abbildung 8: Ungültige Parameteranzahl Primfaktorzerlegung

Bei allen Testfällen wird hier die Fehlermeldung für eine ungültige Parameteranzahl incl. usage ausgegeben.

2.3.2 Nichtnumerischer Parameter

Eingabewerte:

- asdf
- 17jkl

Ausgabe:

```
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_2 asdf
unparsable part of input value: asdf
USAGE: ./uebung1_2 prositiveEvenValue
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_2 17jkl
unparsable part of input value: jkl
USAGE: ./uebung1_2 prositiveEvenValue
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$
```

Abbildung 9: Nichtnumerischer Parameter (Primfaktorzerlegung)

Bei allen Testfällen wird hier der nichtnumerische Teil des Parameters incl. usage ausgegeben.

2.3.3 Ungültiger Wertebereich

Eingabewerte:

- 999999999999

- -999999999999

Ausgabe:

```
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_2 999999999999
parsing of input value failed: Numerical result out of range
USAGE: ./uebung1_2 prositiveEvenValue
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_2 -999999999999
parsing of input value failed: Numerical result out of range
USAGE: ./uebung1_2 prositiveEvenValue
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$
```

Abbildung 10: Ungültiger Wertebereich (Primfaktorzerlegung)

Bei allen Testfällen wird die Meldung, dass der Eingabewert aufgrund der Einschränkungen des Datentyps außerhalb des Wertebereichs liegt, incl. usage ausgegeben.

2.3.4 Einzelner Primfaktor

Eingabe:

- 4096
- 2

Ausgabe:

```
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_2 4096
2^12
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_2 2
2^1
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$
```

Abbildung 11: Einzelner Primfaktor

Hier wird in der Schleife jeweils keine Ausgabe getätigt, es sind daher beide Werte mit jeweils einem Primfaktor abbildbar.

2.4 Mehrere Primfaktoren

- 123456
- 22

Ausgabe:

```
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_2 123456
2^6 * 3^1 * 643^1
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$ ./uebung1_2 22
2^1 * 11^1
ckapplmueller@ubuntu:/mnt/hgfs/share/uebung1$
```

Abbildung 12: Mehrere Primfaktoren

Hier werden in der Schleife bereits Primfaktoren ausgegeben, es sind also mehrere Primfaktoren involviert.

3 1.2 ERRNO & Co

3.1 Fehlende fehlerbehandlung von atoi

Werden der Funktion *atoi* Werte übergeben die nicht numerisch bzw. außerhalb des Wertebereichs sind, so wird dies von *atoi* ignoriert, und folgendes Verhalten gezeigt:

Zahl mit Buchstaben am Ende („12asdf“)	Zahl (12)
Zahl eingeschlossen von Buchstaben („as12df“)	0
Nur Buchstaben („asdf“)	0
Positiver Wert außerhalb des Bereichs („999999999999“)	Maximaler Integerwert (2147483647)
Negativer Wert außerhalb des Bereichs („-999999999999“)	Minimaler Integerwert (-2147483648)

Dieses Verhalten ist auch teilweise in der man-page von *atoi* beschrieben:

> [...]The behavior is the same as *strtol(nptr, NULL, 10);* except that *atoi()* does not detect errors.[...]

Wie in diesem Auszug der man-page ersichtlich, ist *atoi* nichts anderes, als die Abstraktion einer anderen Funktion. Nun liefert diese Funktion aber sowohl werte in *ERRNO* als auch den nicht parsebaren Teil des Eingabeparameters.

Auswertung nicht parsebarer Zeichen

Wie bereits erwähnt, liefert die Funktion *strol* den nicht parsebaren Teil des Eingabeparameters zurück. dies geschieht über eine als 2. Parameter übergebene Adresse eines Sting (character-array). Diese Methode findet im 2. Beispiel (Primfaktorenzerlegung Zeile 27) Anwendung.

3.2 Erörterungen zu ERRNO

ERRNO ist ein Mechanismus zur Fehlererkennung, welcher der C-Standardbibliothek der verwendeten C-Version nach folgende beiden Fehler unterstützen muss:

- *EDOM*

  Wird erkannt, wenn Eingabewerte außerhalb der Domäne einer Funktion liegen übergeben werden.

  z.B. $\sqrt{-1}$, *sqrt(-1)*

- *ERANGE*

  Wird erkannt, wenn Eingabewerte außerhalb des Wertebereichs einer Funktion liegen übergeben werden.

  z.B. *strol(”999999999999”, NULL, 10)*

Der übergebenen Wert liegt außerhalb des gültigen Integer-Bereichs

Um die Integer-Werte der Variable *errno* in eine leicht verständliche Aussage zu übersetzten steht die Funktion *strerror* zur Verfügung, die im 2. Beispiel (Primfaktorenzerlegung, Zeile 31) Anwendung findet.

Um zu gewährleisten, dass die Fehler die via *ERRNO* detektiert werden auch zu der bzw. einer der Funktionen gehört, von der bzw. denen man den Fehlerwert erhalten will, ist es ratsam, vorher mit *errno = 0* den Fehlerwert zurückzusetzten.

Abbildungsverzeichnis

1 Ungültige Parameterzahl (Dreieckszerlegung) 4
2 Ungültiger Wertebereich (Dreieckszerlegung) 4
3 Ungültiges Dreieck 5
4 Rechtwinkeliges Dreieck 6
5 Gleichschenkeliges Dreieck 6
6 Gleichseitiges Dreieck 7
7 Rechtwinkelig-gleichschenkeliges Dreieck 7
8 Ungültige Parameteranzahl Primfaktorzerlegung 10
9 Nichtnumerischer Parameter (Primfaktorzerlegung) 10
10 Ungültiger Wertebereich (Primfaktorzerlegung) 11
11 Einzelner Primfaktor 11
12 Mehrere Primfaktoren 11