



Übung 01

SWO3 WS 2014/15
Gruppe 2

Roman Lumetsberger
1310307026



25. September 2014

Inhaltsverzeichnis

1	Aufgabe 1 - Dreieck - Tester	2
1.1	Lösungsidee	2
1.1.1	Gültigkeit des Dreiecks	2
1.1.2	Gleichseitiges Dreieck	2
1.1.3	Gleichschenkliges Dreieck	2
1.1.4	Rechtwinkliges Dreieck	2
1.1.5	Sonstiges Dreieck	2
1.2	Sourcecode	3
1.3	Testfälle	5
1.3.1	Testfall 1 - ungültige Parameter	5
1.3.2	Testfall 2 - gleichseitig	5
1.3.3	Testfall 3 - gleichschenkl.	5
1.3.4	Testfall 4 - rechtwinklig	6
1.3.5	Testfall 5 - sonstiges Dreieck	6
1.3.6	Testfall 6 - ungültiges Dreieck	7
1.3.7	Testfall 7 - rechtwinklig gleichschenkl.	8
2	Aufgabe 2 - Primfaktorzerlegung	9
2.1	Lösungsidee	9
2.2	Sourcecode	9
2.3	Testfälle	10
2.3.1	Testfall 1 - keine Eingabe	10
2.3.2	Testfall 2 - keine Zahl	10
2.3.3	Testfall 3 - ungültige Eingabe	11
2.3.4	Testfall 4 - erstes gültiges Element	11
2.3.5	Testfall 5 - vorgegebenes Beispiel	11
2.3.6	Testfall 6 - vorgegebenes Beispiel	11
2.3.7	Testfall 7 - vorgegebenes Beispiel	12
2.3.8	Testfall 8 - vorgegebenes Beispiel	12
3	Aufgabe 3 - ERRNO & Co	13
3.1	Lösungsidee	13
3.1.1	Beschreibung ERRNO	13
3.1.2	Beschreibung strtol	13
3.2	Sourcecode	14
3.3	Testfälle	15
3.3.1	Testfall 1 - keine Eingabe	15
3.3.2	Testfall 2 - keine Zahl	15
3.3.3	Testfall 3 - Zahlen und Buchstaben	16
3.3.4	Testfall 4 - Überlauf	16
3.3.5	Testfall 5 - Unterlauf	16

1 Aufgabe 1 - Dreieck - Tester

1.1 Lösungsidee

Das Programm braucht genau 3 Eingabewerte, die geprüft und auf Fließkommawerte umgewandelt werden müssen. Folgende Überprüfungen müssen dann mit den Werten durchgeführt werden:

1.1.1 Gültigkeit des Dreiecks

Dies kann mit **Hinweis 1** geprüft werden, wobei hier alle Kombinationen getestet werden müssen. Es macht aber keinen Unterschied, ob man $a+b \leq c$ oder $b+a \leq c$ betrachtet.

Dadurch ergeben sich genau 3 Fälle:

- $a+b \leq c$
- $a+c \leq b$
- $b+c \leq a$

1.1.2 Gleichseitiges Dreieck

Ein Dreieck ist gleichseitig, wenn alle 3 Seiten gleich sind.

- $a = b = c$

1.1.3 Gleichschenkliges Dreieck

Ein Dreieck ist gleichschenkelig, wenn 2 Seiten gleich sind. Hier ergeben sich wieder 3 Fälle:

- $a = b$
- $a = c$
- $b = c$

1.1.4 Rechtwinkliges Dreieck

Ob ein Dreieck rechtwinklig ist, kann mit dem Satz von Pythagoras überprüft werden. Auch hier gibt es wieder 3 Fälle:

- $a^2 + b^2 = c^2$
- $a^2 + c^2 = b^2$
- $b^2 + c^2 = a^2$

1.1.5 Sonstiges Dreieck

Wenn ein Dreieck gültig ist, aber weder gleichseitig, rechtwinklig oder gleichschenkelig, dann ist es ein sonstiges Dreieck.

1.2 Sourcecode

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <float.h>
4  #include <math.h>
5
6  /* Special epsilon is defined here because DBL_EPSILON is
7     too accurate here to find valid examples */
8  #define EPSILON 0.0000001
9
10 /* defines the bool type */
11 typedef enum bool {false, true} bool;
12
13 /*
14    Checks if the given triangle is valid
15 */
16 bool checkValidTriangle(double a, double b, double c) {
17     if (( a + b ) <= c ) return false;
18     if (( a + c ) <= b ) return false;
19     if (( b + c ) <= a ) return false;
20     return true;
21 }
22
23 /*
24    Checks if the given triangle is orthogonal
25    by using pythagoras
26 */
27 bool checkOrthogonal(double a, double b, double c) {
28     /* the absolute value is checked against the defined EPSILON */
29     if (fabs((a*a + b*b ) - c*c) < EPSILON) return true;
30     if (fabs((a*a + c*c ) - b*b) < EPSILON) return true;
31     if (fabs((b*b + c*c ) - a*a) < EPSILON) return true;
32     return false;
33 }
34
35 /*
36    Checks if the triangle is equilateral
37 */
38 bool checkEquilateral(double a, double b, double c) {
39     /* the absolute value is checked against the defined EPSILON */
40     return fabs(a - b) < EPSILON && fabs(a - c) < EPSILON && fabs(b - c) < EPSILON;
41 }
42
43 /*
44    Checks if the given triangle is equal sided
45 */
46 bool checkEqualSided(double a, double b, double c) {
```

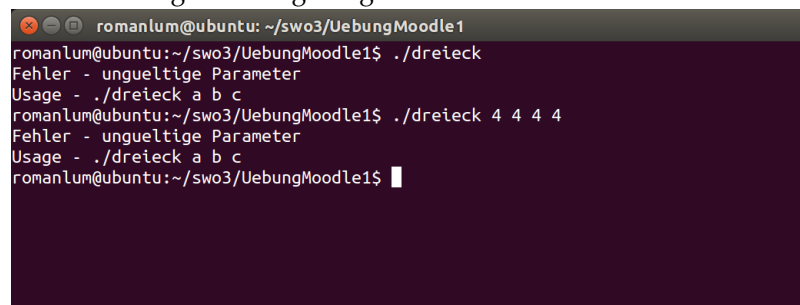
```
47  /* the absolute value is checked against the defined EPSILON */
48  if ( fabs(a - b) < EPSILON ) return true;
49  if ( fabs(a - c) < EPSILON ) return true;
50  if ( fabs(b - c) < EPSILON ) return true;
51  return false;
52 }
53
54 int main(int argc, char *argv[]) {
55     double a,b,c;
56     bool isEqualSided = false;
57     bool isEquilateral = false;
58     bool isOrthogonal = false;
59
60     if ( argc != 4 ) {
61         printf("Fehler - ungueltige Parameter\n");
62         printf("Usage - %s a b c\n", argv[0]);
63         return -1;
64     }
65
66     /* Convert the given parameters to double */
67     a = atof(argv[1]);
68     b = atof(argv[2]);
69     c = atof(argv[3]);
70
71     if ( ! checkValidTriangle(a, b, c) ) {
72         printf("Ergebnis:\nungueltig\n");
73         return EXIT_SUCCESS;
74     }
75
76     isEquilateral = checkEquilateral(a, b, c);
77     /* optimization, equilateral triangles are always equal sided
78     an not orthogonal */
79     if ( isEquilateral ) {
80         isEqualSided = true;
81     }
82     else {
83         isEqualSided = checkEqualSided(a, b, c);
84         isOrthogonal = checkOrthogonal(a, b, c);
85     }
86
87     printf("Ergebnis:\n");
88
89     if( isEqualSided )
90         printf("gleichschenkelig\n");
91
92     if( isEquilateral )
93         printf("gleichseitig\n");
94
95     if( isOrthogonal )
```

```
96     printf("rechtwinklig\n");
97
98     if( !isEqualSided && !isEquilateral && !isOrthogonal)
99         printf("sonstiges Dreieck\n");
100
101     return EXIT_SUCCESS;
102
103 }
```

1.3 Testfälle

1.3.1 Testfall 1 - ungültige Parameter

Erwartetes Ergebnis: ungültig



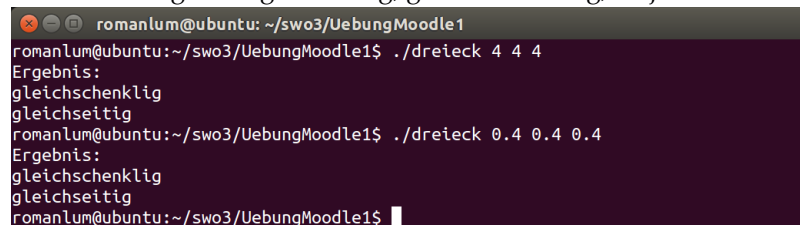
```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck
Fehler - ungeltige Parameter
Usage - ./dreieck a b c
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 4 4 4
Fehler - ungeltige Parameter
Usage - ./dreieck a b c
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

1.3.2 Testfall 2 - gleichseitig

Eingabe: 4 4 4

Eingabe: 0.4 0.4 0.4

Erwartetes Ergebnis: gleichseitig, gleichschenkelig, da ja auch 2 Seiten gleich lang sind



```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 4 4 4
Ergebnis:
gleichschenkelig
gleichseitig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 0.4 0.4 0.4
Ergebnis:
gleichschenkelig
gleichseitig
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

1.3.3 Testfall 3 - gleichschenkelig

Eingabe: 3 3 5

Eingabe: 5 3 3

Eingabe: 3 5 3

Eingabe: 0.3 0.5 0.3

Erwartetes Ergebnis: gleichschenkelig

```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 3 3 5
Ergebnis:
gleichschenkelig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 5 3 3
Ergebnis:
gleichschenkelig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 3 5 3
Ergebnis:
gleichschenkelig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 0.3 0.5 0.3
Ergebnis:
gleichschenkelig
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

1.3.4 Testfall 4 - rechtwinklig

Eingabe: 3 4 5

Eingabe: 5 4 3

Eingabe: 4 3 5

Eingabe: 5 3 4

Eingabe: 3 5 4

Eingabe: 4 5 3

Eingabe: 0.004 0.005 0.003

Erwartetes Ergebnis: rechtwinklig

```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 3 4 5
Ergebnis:
rechtwinklig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 5 4 3
Ergebnis:
rechtwinklig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 4 3 5
Ergebnis:
rechtwinklig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 5 3 4
Ergebnis:
rechtwinklig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 3 5 4
Ergebnis:
rechtwinklig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 4 5 3
Ergebnis:
rechtwinklig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 0.004 0.005 0.003
Ergebnis:
rechtwinklig
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

1.3.5 Testfall 5 - sonstiges Dreieck

Eingabe: 2 3 4

Eingabe: 4 3 2

Eingabe: 3 2 4

Eingabe: 4 2 3

Eingabe: 3 4 2

Eingabe: 2 4 3

Eingabe: 2.2 4.11 3.609

Erwartetes Ergebnis: sonstiges Dreieck

```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 2 3 4
Ergebnis:
sonstiges Dreieck
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 4 3 2
Ergebnis:
sonstiges Dreieck
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 3 2 4
Ergebnis:
sonstiges Dreieck
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 4 2 3
Ergebnis:
sonstiges Dreieck
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 3 4 2
Ergebnis:
sonstiges Dreieck
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 2 4 3
Ergebnis:
sonstiges Dreieck
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 2.2 4.11 3.609
Ergebnis:
sonstiges Dreieck
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

1.3.6 Testfall 6 - ungültiges Dreieck

Eingabe: 3 4 42

Eingabe: 42 4 3

Eingabe: 0.42 1 3.999

Eingabe: 0 0 0

Eingabe: -1 -1 -3

Eingabe: -1 3 2

Eingabe: 0.99 -0.2 2

Erwartetes Ergebnis: ungültig

```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 3 4 42
Ergebnis:
ungueltig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 42 4 3
Ergebnis:
ungueltig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 0.42 1 3.999
Ergebnis:
ungueltig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 0 0 0
Ergebnis:
ungueltig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck -1 -1 -3
Ergebnis:
ungueltig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck -1 3 2
Ergebnis:
ungueltig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 0.99 -0.2 2
Ergebnis:
ungueltig
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

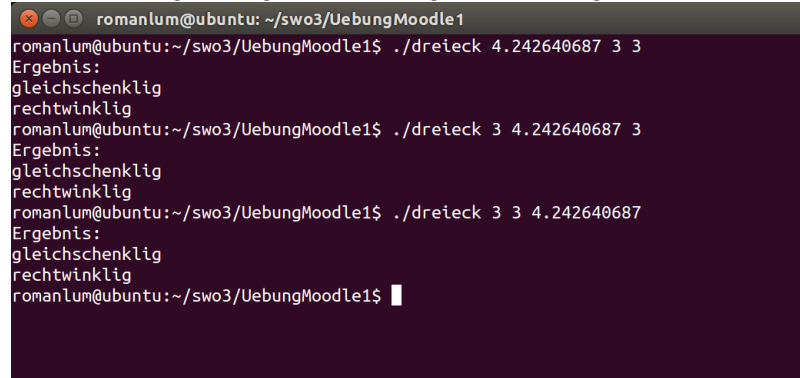

1.3.7 Testfall 7 - rechtwinklig gleichschenkelig

Eingabe: 4.242640687 3 3

Eingabe: 3 4.242640687 3

Eingabe: 3 3 4.242640687

Erwartetes Ergebnis: gleichschenkelig, rechtwinklig



```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 4.242640687 3 3
Ergebnis:
gleichschenkelig
rechtwinklig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 3 4.242640687 3
Ergebnis:
gleichschenkelig
rechtwinklig
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./dreieck 3 3 4.242640687
Ergebnis:
gleichschenkelig
rechtwinklig
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

2 Aufgabe 2 - Primfaktorzerlegung

2.1 Lösungsidee

Das Programm braucht genau einen Eingabewert, der geprüft und auf int konvertiert werden muss. Die Zahl muss größer als 0 sein.

Der Algorithmus läuft dann über alle Teiler ab 2 durch und prüft, ob die Zahl dividiert durch den Teiler 0 Rest ergibt.

- Ist dies der Fall, so wird ein Zähler, der die Anzahl der Teilungen angibt, erhöht und die Zahl durch den Teiler dividiert.
- Ist dies nicht der Fall, dann muss geprüft werden, ob der Zähler größer 0 ist und wenn ja, dann muss der Teiler und Zähler in Exponentialschreibweise ausgegeben werden. Der Zähler wird anschließend auf 0 gesetzt und der Teiler um 1 erhöht.

Ist der Teiler größer als die Zahl selbst, dann kann der Vorgang beendet werden.

Ist der Zähler noch > 0, dann muss auch noch der letzte Teiler ausgegeben werden.

2.2 Sourcecode

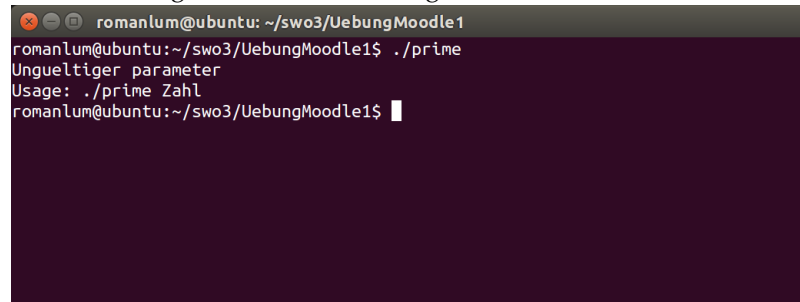
```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[]) {
5     int value, divider, count;
6
7     if(argc != 2) {
8         printf("Ungueltiger parameter\n");
9         printf("Usage: %s Zahl\n", argv[0]);
10        return EXIT_FAILURE;
11    }
12
13    value=atoi(argv[1]);
14    if (value < 1) {
15        printf("Bitte positive, ganze Zahl eingeben.\n");
16        return EXIT_SUCCESS;
17    }
18    else if(value == 1) { /* special case 1, which cannot be splitted into prime factors */
19        printf("2^0\n");
20        return EXIT_SUCCESS;
21    }
22
23    divider = 2;
24    count = 0;
25
26    while (divider <= value) {
27        if ((value % divider) == 0) {
28            count++;
29            value = value / divider;
```

```
30     }
31     else {
32         if (count > 0) {
33             printf("%d~%d * ",divider,count);
34         }
35         divider++;
36         count = 0;
37     }
38 }
39
40 /* display last factor */
41 if(count > 0)
42     printf("%d~%d\n", divider, count);
43
44 return EXIT_SUCCESS;
45 }
```

2.3 Testfälle

2.3.1 Testfall 1 - keine Eingabe

Erwartetes Ergebnis: Fehlermeldung

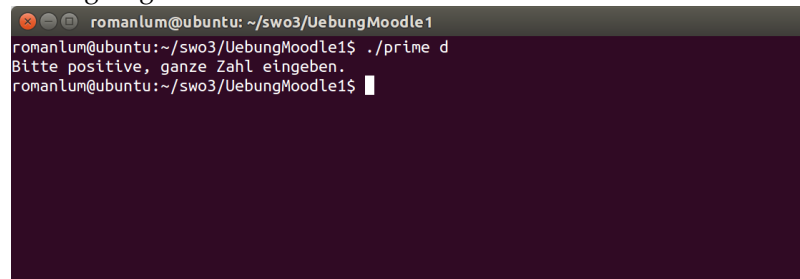


```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./prime
Ungültiger parameter
Usage: ./prime Zahl
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

2.3.2 Testfall 2 - keine Zahl

Eingabe: d

Erwartetes Ergebnis: Meldung, dass eine positive, ganze Zahl eingegeben werden muss, da atoi bei ungültigen Werten 0 liefert



```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./prime d
Bitte positive, ganze Zahl eingeben.
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

2.3.3 Testfall 3 - ungültige Eingabe

Eingabe: 0

Eingabe: -999

Erwartetes Ergebnis: Meldung, dass eine positive, ganze Zahl eingegeben werden muss

```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./prime 0
Bitte positive, ganze Zahl eingeben.
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./prime -999
Bitte positive, ganze Zahl eingeben.
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

2.3.4 Testfall 4 - erstes gültiges Element

Eingabe: 1

Erwartetes Ergebnis: 2^0

```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./prime 1
2^0
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

2.3.5 Testfall 5 - vorgegebenes Beispiel

Eingabe: 10

Erwartetes Ergebnis: $2^1 * 5^1$

```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./prime 10
2^1 * 5^1
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

2.3.6 Testfall 6 - vorgegebenes Beispiel

Eingabe: 256

Erwartetes Ergebnis: 2^8

```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./prime 256
2^8
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

2.3.7 Testfall 7 - vorgegebenes Beispiel

Eingabe: 6534

Erwartetes Ergebnis: $2^1 * 3^3 * 11^2$

```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./prime 6534
2^1 * 3^3 * 11^2
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

2.3.8 Testfall 8 - vorgegebenes Beispiel

Eingabe: 13332

Erwartetes Ergebnis: $2^2 * 3^1 * 11^1 * 101^1$

```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./prime 13332
2^2 * 3^1 * 11^1 * 101^1
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

3 Aufgabe 3 - ERRNO & Co

3.1 Lösungsidee

Die man-page von `atoi` erklärt, dass **`strtol`** als Alternative von `atoi` verwendet werden kann. Diese Methode funktioniert grundsätzlich wie `atoi`, hat aber eine Fehlererkennung. Diese Methode setzt hierzu die Variable **`errno`**, die in der Datei `errno.h` definiert ist.

3.1.1 Beschreibung ERRNO

Diese Variable ist in der C-Standardbibliothek `errno.h` definiert und kann verwendet werden, um Fehler, die bei einem vorherigen Funktionsaufruf aufgetreten sind, auszuwerten. Dabei ist der Wert nur dann aussagekräftig, wenn die aufgerufene Funktion signalisiert, dass ein Fehler aufgetreten ist.

Bei **`strtol`** ist dieser Rückgabewert 0.

Wichtig ist, dass die Variable vor dem Aufruf auf 0 gesetzt und direkt nach dem Aufruf ausgewertet wird. In der C-Standardbibliothek werden verschiedene Fehlercodes definiert, die in der man-page nachgelesen werden können.

3.1.2 Beschreibung `strtol`

Diese Funktion konvertiert eine Zeichenkette in einen `int`.

```
long int strtol(const char *nptr, char **endptr, int base);
```

Dabei ist zu beachten, dass die richtige Basis angegeben wird. In diesem Beispiel 10, für Dezimal.

Diese Funktion setzt bei einem Fehler die Variable `errno` und liefert dann als Rückgabewert 0. Weiters liefert die Funktion einen Zeiger auf eine Zeichenkette `endptr` zurück, die auf das erste ungültige Zeichen der Eingabe zeigt.

- Ist dieser Zeiger gleich mit der Eingabe ==> ganze Zeichenkette ungültig
- Ist das Zeichen an der Zeigerposition `'\0'` ==> ganze Zeichenkette gültig

3.2 Sourcecode

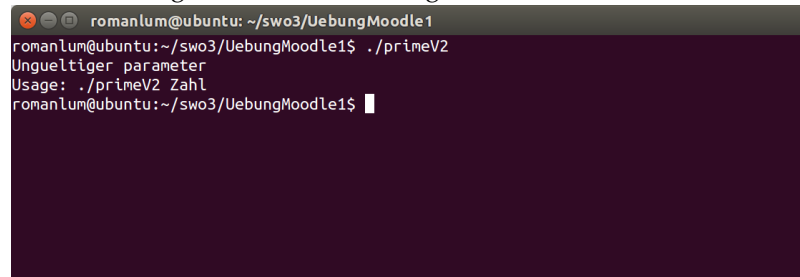
```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <errno.h>
4  #include <limits.h>
5
6  int main(int argc, char *argv[]) {
7      int value, divider, count;
8      char *endptr;
9
10     if(argc != 2) {
11         printf("Ungueltiger parameter\n");
12         printf("Usage: %s Zahl\n", argv[0]);
13         return EXIT_FAILURE;
14     }
15
16     endptr = NULL;
17     errno = 0; /* Reset errno */
18     value = strtol(argv[1], &endptr, 10); /* base = 10 ==> decimal */
19
20     /* Check for errors */
21     if ((errno == ERANGE && (value == LONG_MAX || value == LONG_MIN))) {
22         printf("Fehler Zahlenbereich - Bitte ganze Zahl eingeben die > 0 ist\n");
23         return EXIT_FAILURE;
24     }
25     else if (errno != 0 && value == 0) {
26         printf("Fehler - Bitte ganze Zahl eingeben die > 0 ist\n");
27         return EXIT_FAILURE;
28     }
29
30     /* complete or partial string was invalid */
31     if (endptr == argv[1] || *endptr != '\0') {
32         printf("Bitte positive, ganze Zahl eingeben\n");
33         return EXIT_FAILURE;
34     }
35
36     if (value < 1) {
37         printf("Zahl zu klein: Bitte ganze Zahl eingeben die > 0 ist\n");
38         return EXIT_SUCCESS;
39     }
40     else if (value == 1) { /* special case */
41         printf("2^0\n");
42         return EXIT_SUCCESS;
43     }
44
45     divider = 2;
46     count = 0;
```

```
47
48 while (devider <= value) {
49     if ((value % devider) == 0) {
50         count++;
51         value = value / devider;
52     }
53     else {
54         if (count > 0) {
55             printf("%d~%d * ",devider,count);
56         }
57
58         devider++;
59         count = 0;
60     }
61 }
62 }
63
64 /* display last factor */
65 if(count > 0)
66     printf("%d~%d\n", devider, count);
67
68 return EXIT_SUCCESS;
69 }
```

3.3 Testfälle

3.3.1 Testfall 1 - keine Eingabe

Erwartetes Ergebnis: Fehlermeldung



```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./primeV2
Ungueltiger parameter
Usage: ./primeV2 Zahl
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

3.3.2 Testfall 2 - keine Zahl

Eingabe: d

Eingabe: hallo

Eingabe: zahl

Erwartetes Ergebnis: Fehlermeldung


```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./primeV2 d
Bitte positive, ganze Zahl eingeben
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./primeV2 hallo
Bitte positive, ganze Zahl eingeben
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./primeV2 zahl
Bitte positive, ganze Zahl eingeben
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

3.3.3 Testfall 3 - Zahlen und Buchstaben

Eingabe: 17d

Eingabe: d17

Erwartetes Ergebnis: Fehlermeldung

```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./primeV2 17d
Bitte positive, ganze Zahl eingeben
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./primeV2 d17
Bitte positive, ganze Zahl eingeben
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

3.3.4 Testfall 4 - Überlauf

Eingabe: 9999999999

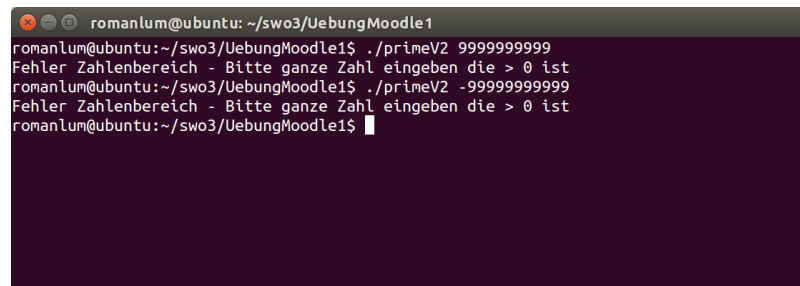
Erwartetes Ergebnis: Fehlermeldung

```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./primeV2 9999999999
Fehler Zahlenbereich - Bitte ganze Zahl eingeben die > 0 ist
romanlum@ubuntu:~/swo3/UebungMoodle1$
```

3.3.5 Testfall 5 - Unterlauf

Eingabe: -9999999999

Erwartetes Ergebnis: Fehlermeldung



```
romanlum@ubuntu: ~/swo3/UebungMoodle1
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./primeV2 9999999999
Fehler Zahlenbereich - Bitte ganze Zahl eingeben die > 0 ist
romanlum@ubuntu:~/swo3/UebungMoodle1$ ./primeV2 -99999999999
Fehler Zahlenbereich - Bitte ganze Zahl eingeben die > 0 ist
romanlum@ubuntu:~/swo3/UebungMoodle1$
```