1 Allgemein

Dieser Teil des Dokuments beinhaltet die Dokumentation der allgemeinen Programmstruktur, die alle Aufgabenstellungen betreffen.

1.1 Lösungsidee

Folgend ist die Lösungsidee der Programmstruktur dokumentiert.
Um nicht alle Aufgabenstellungen in ein riesiges Hauptprogram zu packen, oder die Programme so voneinander so zu trennen, das einzelne Binaries entstehen, soll das Programm so aufgebaut sein, dass alle Teilprogramme sich über ein Hauptprogram aufrufen lassen, wobei das zu startende Teilprogramm über den ersten Parameter beim Ausführen des Programms definiert wird.

Das Programm soll wie folgt aufgeteilt werden:

- header/*.h
  Alle Headerdateien der Subprogramme und des Hauptprogramms
- source/*.c
  Der Source aller Subprogramme
- test/*.sh
  Die Shell Skripten für die Tests
- test/bin/
  Das Verzeichnis in dem das kompilierte Programm für die Test gehalten wird.
- doc/
  Das Verzeichnis in dem die Aufgabenstellung sowie die Dokumentation gehalten wird.
- main.c
  Das Hauptprogramm über welches die Subprogramme aufgerufen werden.

Das gewünschte Subprogramm wird über das Hauptprogramm wie folgt aufgerufen:

```
./main.bin <application_tpe> <application_parameters>
```

Applikationstypen:

- **1** (Dreiecksvalidierung)
- **2** (Primfaktorenzerlegung)
- **3** (Primfaktorenzerlegung errno)

Folgend ist beschrieben wie die Testskripte ausgeführt werden müssen und auch wie das Programm aufgerufen wird.

Ausführen der Testskripten:

1. Entpacken Sie das Archiv auf einen Linux System
2. Navigieren Sie in das Verzeichnis, wo das Shell Script enthalten ist. *(<archive_root>/test)*
3. Stellen Sie sicher das der Benutzer ausreichende Rechte für das entpackte Verzeichnis und seinen Inhalt hat
4. Machen Sie die Skripten ausführbar. `chmod +x *_tests.sh`
5. Führen Sie die Testskripten aus. Das aktuelle Verzeichnis muss *<archive_root>/test* sein, da die Tests sonst nicht funktionieren, da relative Pfade enthalten sind.

Kompilieren des Programms:

Da das Programm in mehrere Sources und Header Dateien aufgeteilt wurde muss das Programm wie folgt kompiliert werden. Dieses Beispiel zeigt die Kompilierung so wie sie in den Tests erfolgt. Je nachdem in welchen Verzeichnis man sich befindet müssen die relativen Pfade angepasst werden.

```
gcc -g -I../../Commons/header -I../header/ ../main.c ../source/triangle.c ../source/prime.c -o bin/main.bin –lm
```

1.2 Implementierung

Folgen ist die Implementierung des Hauptprogramms angeführt, über welches die Teilprogramme gestartet werden.

1.2.1 Main.h

Diese Header Datei spezifiziert die Enumeration der Applikationstypen, welche über das Hauptprogramm aufgerufen werden können.

```
/*
 * main.h
 *
 *  This is the header file for the main program which starts the sub programs.
 *  It bundles the includes and definitions used by the main program.
 *
 *  Created on: Sep 20, 2014
 *      Author: cchet
 */
#ifndef MAIN_H
#define MAIN_H
/* enumeration for the different applications */
typedef enum eApplication {
    triangle = 1, prime = 2, prime_errno = 3
} eApplication;

#endif /* MAIN_H_ */
```

1.2.2 Common.h

Diese Header Datei enthält die Typdefinition von bool und soll in Zukunft wiederverwendet werden und alle gemeinsamen und oft vorkommenden Typen enthalten.

```
/*
 * common.h
 *
 *  Created on: Sep 20, 2014
 *      Author: cchet
 */

#ifndef COMMON_H
#define COMMON_H

/* allows to use bool in c */
typedef enum bool {
    false = 0, true = 1
} bool;

#endif /* COMMON_H_ */
```

1.2.3 Main.c

Diese Datei beinhaltet den Source für das Hauptprogram.

```
/*
 * main.c
 *
 * This represents the source of the main program which delegates to the sub program
 * depending on the given application type.
 *
 *  Created on: Sep 18, 2014
 *      Author: cchet
 *
 */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "main.h"
#include "triangle.h"
#include "prime.h"

int main(int argc, char** argv) {
    int application_type;
    if (argc < 2) {
        printf("No arguments are given !!!\n");
        printf("./main.h <application_type> <application_parameters>\n");
        return EXIT_FAILURE;
    } else {
        application_type = atoi(argv[1]);
        /* Triangle validator */
        if (application_type == triangle) {
            return ValidateTriangle(argc, argv);
        }
        /* prime factorizer */
        else if (application_type == prime) {
            return ValidateNumber(argc, argv);
        }
        /* prime factorizer error aware */
        else if (application_type == prime_errno) {
            return ValidateNumberErrorAware(argc, argv);
        }
        /* unknown application type */
        else {
            printf("Application type is unknown !!! application_type: %s\n",
                    argv[1]);
            return EXIT_FAILURE;
        } /* if */
    } /* if */
}
```

2 Dreiecks Tester

Der folgende Teil dieser Dokumentation behandelt die Aufgabenstellung Dreieckstester, welche verlangt, dass drei gegebene Seitenlängen geprüft werden, ob diese ein gültiges Dreieck bilden. Des Weiteren soll der Typ des Dreiecks bestimmt werden (Gleichseitig, Gleichschenklig, Rechtwinklig, Normale).

2.1 Lösungsidee

Das Programm soll Parameter des Typs Integer und Double (aus Sicht des Aufrufers) unterstützen. Bevor diese verarbeitet werden, sollen diese aus dem Datentyp String in den Datentyp Double konvertiert werden. (Bsp.: ‚1', ‚2.5')
Fehlerhafte Aufrufe sollen behandelt und dem Aufrufer mitgeteilt werden.

Folgende Fehler sind zu erwarten:

1. Keine oder unvollständige Parameter
2. Parameter haben ungültige Werte (param <= 0, param repräsentiert keinen numerischen Wert bzw. kann nicht in einen konvertiert werden).
3. Angegebenen Seitenlängen repräsentieren kein gültige Dreieck.

Die verschiedenen Typen der Dreieckstypen sollen wie folgt ermittelt werden:

1. Gleichseitiges Dreieck
   a = b = c
2. Gleichschenkliges Dreieck
   (a = b) || (b = c) || (c = a)
3. Rechtwinkliges Dreieck
   a. Ermittlung der Hypotenuse, Ankathete, Gegenkathete
   b. Berechnung der Gegenkathete^2 mit c^2 = a^2 + b^2;
   c. Vergleichen ob die gegebene Gegenkathete^2 = berechnete Gegenkathete^2
4. Jedes weitere Dreieck mit gültigen Seitenlängen ist ein normales Dreieck.

Da bekannt ist, dass Double Ungenauigkeitsfehler aufweist, ist damit zu rechnen, dass bei Angabe von Dezimalwerten Fehler zu erwarten sind.

Diese Applikation garantiert nicht das Dreiecke mit Dezimalwerten korrekt funktionieren, da Ungenauigkeitsfehler vom Datentyp double die Dreiecke verfälschen können.
Ebenso sei die Fehlertoleranz der Funktion atoi angemerkt, die es erlaubt Argumente wie z.B.: 4_adfdf zu einem Double 4.0 zu konvertieren.

2.2 Implementierung

Der folgende Teil dieser Dokumentation behandelt den implementierten Source, sowie auch Test Source.

2.2.1 Triangle.h

Diese Header Datei enthält alle Funktionen, die für die Dreiecksvalidierung benötigt werden.

```
/*
 * triangle.h
 *
 *  This header files specifies the types and functions used by the triangle sub-program.
 *
 *  Created on: Sep 20, 2014
 *      Author: cchet
 */

/* User libs */
#include "common.h"
#ifndef TRIANGLE_H_
#define TRIANGLE_H

/*
 * Specifies the types which the given triangle can be of
 */
typedef enum eTriangleType {
    equilateral, isosceles, square, normal, invalid
} eTriangleType;

/**
 * This function represents the main entry point in this sub program, called by the main
program.
 * Please note that triangles defined by double side lengths are not guaranteed to be detected
correctly,
 * because of the fact that there could occur precision errors.
 * This is an restriction of this program.
 *
 * int argc: the argument count
 * char* argv: the given arguments
 */
int ValidateTriangle(int argc, char** argv);

/**
 * Validates the given arguments if they represent a valid double value.
 * The function 'atof' is used which returns 0 when an invalid parameter has been found.
 * This can be used here because a 0 side length is not allowed for valid triangles.
 *
 * argv: the pointer to the arguments array
 *
 * return: true if all arguments are valid doubles in the correct range
 */
bool ContainsInvalidateParams(char** argv);

/*
 * Gets the triangle type which is defined in the enumeration type "eTriangleType".
 *
 * double a: the a side of the triangle
 * double b: the b side of the triangle
 * double c: the c side of the triangle
 *
 * return eTriangleType: the triangle type
 *
 * see: typedef enum eTriangleType {...}
 */
eTriangleType GetTriangleType(double a, double b, double c);

/**
 * Answers the question if the given triangle side lengths representing a square triangle.
 *
 * double a: the a side of the triangle
 * double b: the b side of the triangle
 * double c: the c side of the triangle
```

```
 *
 * return bool: true if the triangle is a square triangle
 */
bool IsSquareTirangle(double a, double b, double c);

/**
 * Analyzes the determined triangle type and returns the proper return value
 * and prints a message to the console.
 *
 * eTriangleType type: the determined triangle type.
 * return int: 0-if valid triangle, 1-if not
 */
int GetResult(enum eTriangleType type);

#endif /* TRIANGLE_H_ */
```

2.2.2 triangle.c

Folgend ist der Source des Subprogramms Dreiecks Tester angeführt.

```
/*
 * triangle validator.c
 *
 * This program is used to validate three given site lengths of an triangle.
 * It is validated if this triangle is an valid triangle and of which type.
 *
 * !!! Note that this program does not guarantee that site lengths with more then precision 2
are correctly handled  !!!
 * !!! This is caused by the usage of double values which are converted by "aotf" function
which has rounding issues !!!
 * !!! E.g.: 2.65 will be converted to double like 2.64999999999999999998
!!!
 * !!! Be aware of that and use this program with just a single precisions such as 2.5
!!!
 *
 *  Created on: Sep 18, 2014
 *      Author: cchet
 *
 */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "triangle.h"

/**
 * The main function of the program
 */
int ValidateTriangle(int argc, char** argv) {

    /* Check for correct argument count */
    if (argc != 5) {
        printf("Exact 3 integer/double parameter must be given !!! count: %i",
                (argc - 2));
        return EXIT_FAILURE;
    } /* if */

    /* Validate for valid double argument */
    if (ContainsInvalidateParams(argv) == true) {
        printf(
                "The arguments are not valid integer/double !!! p1: %s | p2: %s | p3: %s",
                argv[2], argv[3], argv[4]);
        return EXIT_FAILURE;
    } /* if */

    printf(
            "Please note that the validation of a triangle could fail because of the fact\n");
    printf(
            "that double has precision failures during parsing regarding backed floating point
!!!!\n");
    printf("Entered values: p1: %f | p2: %f | p3: %f\n", atof(argv[2]),
            atof(argv[3]), atof(argv[4]));

    return GetResult(
            GetTriangleType(atof(argv[2]), atof(argv[3]), atof(argv[4])));
```

```
}

bool ContainsInvalidateParams(char** argv) {
    return ((atof(argv[2]) <= 0) || (atof(argv[3]) <= 0) || (atof(argv[4]) <= 0));
}

eTriangleType GetTriangleType(double a, double b, double c) {
    if ((a + b <= c) || (a + c <= b) || (b + c <= a)) {
        return invalid;
    } else if ((a == b) && (b == c)) {
        return equilateral;
    } else if ((a == b) || (a == c) || (b == c)) {
        return isosceles;
    } else if (IsSquareTirangle(a, b, c)) {
        return square;
    } else {
        return normal;
    } /* if */
}

bool IsSquareTirangle(double a, double b, double c) {
    double coordinates[3] = { [0]=a, [1]= b, [2]= c };
    int i;
    double hypotenuse = -1;
    double adjacentSide = coordinates[0];
    double oppositeSite = -1;

    /* Determine the the hypotenuse and adjancentSide */
    for (i = 0; i < 3; ++i) {
        if (coordinates[i] <= adjacentSide) {
            adjacentSide = coordinates[i];
        }
        else if (coordinates[i] > hypotenuse) {
            hypotenuse = coordinates[i];
        } /* if */
    } /* for */

    /* Determine the oppositeSideIndex*/
    for (i = 0; i < 3; ++i) {
        if ((coordinates[i] != adjacentSide)
                && (coordinates[i] != hypotenuse)) {
            oppositeSite = coordinates[i];
        } /* if */
    } /* for */

    /* Try to calculate the oppositeSideIndex via c^2 = root(a^2 + b^2) and compare to given
length of site*/
    return ((pow(adjacentSide, 2) + pow(oppositeSite, 2)) == pow(hypotenuse, 2)) ?
            true : false;
}

int GetResult(enum eTriangleType type) {
    switch (type) {
    case invalid:
        printf(
                "our given triangle sites represent a 'invalid' triangle Please check them and
try again !!!");
        return EXIT_FAILURE;
        break;
    case equilateral:
        printf("Your given triangle sites represent a 'equilateral' triangle");
        return EXIT_SUCCESS;
        break;
    case isosceles:
        printf("Your given triangle sites represent a 'isosceles' triangle");
        return EXIT_SUCCESS;
        break;
    case square:
        printf("Your given triangle sites represent a 'square' triangle");
        return EXIT_SUCCESS;
        break;
    case normal:
        printf("Your given triangle sites represent a 'normal' triangle");
        return EXIT_SUCCESS;
        break;
    default:
```

```
        printf(
                "This should not happen, because this would mean that not all enums are
handled by this switch case");
        return EXIT_FAILURE;
        break;
    } /* case */

    /* Should never get here */
    printf("Switch case did not take place !!!");
    return EXIT_FAILURE;
}
```

2.2.3 Shell Script – Tests

Dieses Shell Skript realisiert die Tests des Subprogramms Dreiecks Tester.

```
#!/bin/sh
## Varaibles used during build and tests ##
APPLICATION_TYPE=1
SOURCE_MAIN=../main.c
BINARY_MAIN=bin/main_trianlge_test.bin
SOURCE_TRIANGLE=../source/triangle.c
SOURCE_PRIME=../source/prime.c
INCLUDE_HEADERS_COMMONS=../../Commons/header
INCLUDE_HEADERS=../header/

echo "################################################################################"
echo "These are the tests for the triangle_validator"
echo "################################################################################ \n"

## Compile the source file ##
## build triangle source
echo "compiling source file '$SOURCE_MAIN' and dependencies to target $BINARY_MAIN \n"
echo "$ gcc -g -I$INCLUDE HEADERS COMMONS -I$INCLUDE HEADERS $SOURCE MAIN $SOURCE TRIANGLE
$SOURCE_PRIME -o $BINARY MAIN -lm\n"
gcc -g -I$INCLUDE_HEADERS_COMMONS -I$INCLUDE_HEADERS $SOURCE_MAIN $SOURCE_TRIANGLE
$SOURCE_PRIME -o $BINARY_MAIN -lm

## Check if file has been compiled
if [ -f "$BINARY_MAIN" ]
then
    echo "source compiled starting tests now ...\n"
else
    echo "compile of $SOURCE_MAIN failed\n"
    exit 1
fi

################################################################
## Test wrong parameter count                                 ##
################################################################
echo "--------------------------------------------------------------------------------"
echo "Testing worng parameter count"
echo "--------------------------------------------------------------------------------"
## Case 1
echo "Case 1: no param"
$BINARY_MAIN $APPLICATION_TYPE
echo "\n"

## Case 2
echo "Case 2: 1 param"
$BINARY_MAIN $APPLICATION_TYPE 1.0
echo "\n"

## Case 3
echo "Case 3: 2 param"
$BINARY_MAIN $APPLICATION_TYPE 1.0 2.0
echo "\n"

## Case 4
echo "Case 4: 4 param"
$BINARY_MAIN $APPLICATION_TYPE 1.0 2.0 2 2
echo "\n"

## Case 5
echo "Case 5: 5 param"
$BINARY_MAIN $APPLICATION_TYPE 1.0 2.0 2 2 3
```

```
echo "\n"

#################################################################
## Test invalid triangle site lengths                          ##
#################################################################
echo "-------------------------------------------------------------------------------"
echo "Testing with '0' parameter "
echo "-------------------------------------------------------------------------------"
## Case 1
echo "Case 1: a = 0 | b = 1 | c = 2"
$BINARY_MAIN $APPLICATION_TYPE 0 1 2
echo "\n"

## Case 2
echo "Case 3: a = 1 | b = 0 | c = 2"
$BINARY_MAIN $APPLICATION_TYPE 1 0 2
echo "\n"

## Case 3
echo "Case 4: a = 1 | b = 2 | c = 0"
$BINARY_MAIN $APPLICATION_TYPE 1 2 0
echo "\n"

echo "-------------------------------------------------------------------------------"
echo "Testing with character parameter  "
echo "-------------------------------------------------------------------------------"
## Case 1
echo "Case 1: a = abc | b = 1 | c = 2"
$BINARY_MAIN $APPLICATION_TYPE abc 1 2
echo "\n"

## Case 2
echo "Case 2: a = 1 | b = abc | c = 2"
$BINARY_MAIN $APPLICATION_TYPE 1 abc 2
echo "\n"

## Case 3
echo "Case 3: a = 1 | b = 2 | c = abc"
$BINARY_MAIN $APPLICATION_TYPE 1 2 abc
echo "\n"

## Case 4
echo "Case 4: a = 1 | b = 2 | c = asdfadsfdf_454544"
$BINARY_MAIN $APPLICATION_TYPE 1 2 asdfadsfdf_454544
echo "\n"

## Case 5
echo "Case 5: a = 1 | b = asdfadsfdf_454544 | c = 2"
$BINARY_MAIN $APPLICATION_TYPE 1 asdfadsfdf_4545442 2
echo "\n"

## Case 6
echo "Case 6: a = asdfadsfdf_454544 | b = 1 | c = 2"
$BINARY_MAIN $APPLICATION_TYPE asdfadsfdf_454544 1 2
echo "\n"

echo "-------------------------------------------------------------------------------"
echo "Testing with invalid site lengths "
echo "-------------------------------------------------------------------------------"
## Case 1
echo "Case 1: wrong dimensions of square triangle (parese issue of 'atof'): a = 3.41 | b = 6.0
| c = 6.9"
echo "         Program does not support double precision greather than 1 and do not check this"
$BINARY_MAIN $APPLICATION_TYPE 3.41 6.0 6.9
echo "\n"

## Case 2
echo "Case 2: wrong dimensions of triangle: a = 2.5 | b = 2.5 | c = 5.0"
$BINARY_MAIN $APPLICATION_TYPE 2.5 2.5 5.0
echo "\n"

echo "-------------------------------------------------------------------------------"
echo "Testing with valid site lengths "
echo "-------------------------------------------------------------------------------"
## Case 1
echo "Case 1: 3 equilateral triangle: a = 2.5 | b = 2.5 | c = 2.5"
```

```
$BINARY_MAIN $APPLICATION_TYPE 2.5 2.5 2.5
echo "\n"

## Case 2
echo "Case 2: isosceles triangle: a = 2.5 | b = 2.5 | c = 3.54"
$BINARY_MAIN $APPLICATION_TYPE 2.5 2.5 3.54
echo "\n"

## Case 3
echo "Case 3: normal triangle:  a = 2.18 | b = 4.5 | c = 5"
$BINARY_MAIN $APPLICATION_TYPE 2.18 4.5 5
echo "\n"

## Case 4
echo "Case 4: square triangle: a = 6 | b = 4.5 | c = 7.5"
$BINARY_MAIN $APPLICATION_TYPE 6 4.5 7.5
echo "\n"

## Case 5
echo "Case 5: square triangle: a = 6_asasas | b = 4.5 | c = 7.5"
$BINARY_MAIN $APPLICATION_TYPE 6_asas 4.5 7.5
echo "\n"

## Case 6
echo "Case 6: square triangle: a = 6 | b = 4.5_asasas | c = 7.5"
$BINARY_MAIN $APPLICATION_TYPE 6 4.5_asasas 7.5
echo "\n"

## Case 7
echo "Case 7: square triangle: a = 6 | b = 4.5 | c = 7.5_asasas"
$BINARY_MAIN $APPLICATION_TYPE 6 4.5 7.5_asasas
echo "\n"

## Cleanup binary
echo "tests done !!!!!"
echo "Remove binaries ..."
rm -rf $BINARY_MAIN
echo "Binaries removed"
exit 0
```

2.3 Tests

Der folgende Teil dieser Dokumentation behandelt die Testfälle für dieses C-Programm.

Die Tests wurden im oben angeführten Shell Skript implementiert und müssen wie im Punkt **1. Allgemein** beschrieben ausgeführt werden.

Es wurde versucht die Ungenauigkeitsfehler vom Datentyp double wie folgt zu verhindern, jedoch gelang mir dies nicht.

```
double power = pow(POT_TEN, precision);
double r = value * power;
double t1 = floor(r);
double t2 = t1 / pow(POT_TEN, precision);
return t2;
```

Der beseitigte Ungenauigkeitsfehler trat aber nach der Division wieder auf.

Bsp.: 7.546 hatte eine Ungenauigkeit von 0.0000000000000003, welche sich der Division wieder erneut gezeigt hat.

Da meine Erfahrungen mit C noch nicht ausreichen, konnte dieser Fehler nicht behoben werden.

2.3.1 Testfälle Anzahl Parameter

Diese Tests testen die Fehlerbehandlung von fehlenden Parametern.

```
--------------------------------------------------------------------------------
Testing worng parameter count
--------------------------------------------------------------------------------
Case 1: no param
Exact 3 integer/double parameter must be given !!! count: 0

Case 2: 1 param
Exact 3 integer/double parameter must be given !!! count: 1

Case 3: 2 param
Exact 3 integer/double parameter must be given !!! count: 2

Case 4: 4 param
Exact 3 integer/double parameter must be given !!! count: 4

Case 5: 5 param
Exact 3 integer/double parameter must be given !!! count: 5
```

Bei einer Angabe von einer falschen Anzahl von Parametern wird eine entsprechende Fehlermeldung ausgegeben, die anzeigt wie viele Parameter verlangt werden und wie viele angegeben wurden.

2.3.2 Testfälle ungültige Parameter

Diese Tests testen die Fehlerbehandlung wenn die Parameter keine gültigen Dezimalwerte darstellen.

```
--------------------------------------------------------------------------------
Testing with '0' parameter
--------------------------------------------------------------------------------
Case 1: a = 0 | b = 1 | c = 2
The arguments are not valid integer/double !!! p1: 0 | p2: 1 | p3: 2

Case 3: a = 1 | b = 0 | c = 2
The arguments are not valid integer/double !!! p1: 1 | p2: 0 | p3: 2

Case 4: a = 1 | b = 2 | c = 0
The arguments are not valid integer/double !!! p1: 1 | p2: 2 | p3: 0
```

Wenn eine Seitenlänge mit 0 angegeben wird, so wird das als Fehler interpretiert, da keine Seitenlänge eines Dreiecks 0 sein kann.

```
--------------------------------------------------------------------------------
Testing with character parameter
--------------------------------------------------------------------------------
Case 1: a = abc | b = 1 | c = 2
The arguments are not valid integer/double !!! p1: abc | p2: 1 | p3: 2

Case 2: a = 1 | b = abc | c = 2
The arguments are not valid integer/double !!! p1: 1 | p2: abc | p3: 2

Case 3: a = 1 | b = 2 | c = abc
The arguments are not valid integer/double !!! p1: 1 | p2: 2 | p3: abc

Case 4: a = 1 | b = 2 | c = asdfadsfdf_454544
The arguments are not valid integer/double !!! p1: 1 | p2: 2 | p3: asdfadsfdf_454544

Case 5: a = 1 | b = asdfadsfdf_454544 | c = 2
The arguments are not valid integer/double !!! p1: 1 | p2: asdfadsfdf_4545442 | p3: 2

Case 6: a = asdfadsfdf_454544 | b = 1 | c = 2
The arguments are not valid integer/double !!! p1: asdfadsfdf_454544 | p2: 1 | p3: 2
```

Ebenfalls müssen alle Parameter sich auf Double Datentypen parsen lassen, was in diesem Fall nicht möglich ist.

2.3.3 Testfälle ungültige Seitenlängen

Diese Tests testen die Fehlerbehandlung wenn die angegebenen Seitenlängen kein gültiges Dreieck darstellen.

```
--------------------------------------------------------------------------------
Testing with invalid site lengths
--------------------------------------------------------------------------------
Case 1: wrong dimensions of square triangle (parese issue of 'atof'): a = 3.41 | b = 6.0 | c = 6.9
        Program does not support double precision greather than 1 and do not check this
Please note that the validation of a triangle could fail because of the fact
that double has precision failures during parsing regarding backed floating point !!!!
Entered values: p1: 3.410000 | p2: 6.000000 | p3: 6.900000
Your given triangle sites represent a 'normal' triangle

Case 2: wrong dimensions of triangle: a = 2.5 | b = 2.5 | c = 5.0
Please note that the validation of a triangle could fail because of the fact
that double has precision failures during parsing regarding backed floating point !!!!
Entered values: p1: 2.500000 | p2: 2.500000 | p3: 5.000000
our given triangle sites represent a 'invalid' triangle Please check them and try again !!!
```

Aufgrund von Ungenauigkeitsfehlern bei dem Parsen auf den Datentyp double können manche Seitenlängen nicht korrekt validiert werden, obwohl diese gültige Seitenlängen eines Dreiecks sind.

Case 1:

Es viel auf das diese Ungenauigkeitsfehler sich hauptsächlich bei mehr als einer Dezimalstelle zeigen. Dies wurde bei der Lösungsidee und auch im Programm dokumentiert, und ebenfalls wird eine Meldung ausgegeben, die dem Aufrufer dies mitteilt, das es bei Dezimalwerten zu Ungenauigkeiten kommen kann und daher eine Validierung fehlschlagen kann. In diesem Fall sollte es ein square Triangel sein, wird aber als normales erkannt, da die Ungenauigkeitsfehler sich niederschlagen.

2.3.4 Testfälle ohne Fehler

Diese Tests testen die Funktionalität des Programms ob alle Dreieckstypen korrekt erkannt werden.

```
--------------------------------------------------------------------------------
Testing with valid site lengths
--------------------------------------------------------------------------------
Case 1: 3 equilateral triangle: a = 2.5 | b = 2.5 | c = 2.5
Please note that the validation of a triangle could fail because of the fact
that double has precision failures during parsing regarding backed floating point !!!!
Entered values: p1: 2.500000 | p2: 2.500000 | p3: 2.500000
Your given triangle sites represent a 'equilateral' triangle

Case 2: isosceles triangle: a = 2.5 | b = 2.5 | c = 3.54
Please note that the validation of a triangle could fail because of the fact
that double has precision failures during parsing regarding backed floating point !!!!
Entered values: p1: 2.500000 | p2: 2.500000 | p3: 3.540000
Your given triangle sites represent a 'isosceles' triangle

Case 3: normal triangle:  a = 2.18 | b = 4.5 | c = 5
Please note that the validation of a triangle could fail because of the fact
that double has precision failures during parsing regarding backed floating point !!!!
Entered values: p1: 2.180000 | p2: 4.500000 | p3: 5.000000
Your given triangle sites represent a 'normal' triangle

Case 4: square triangle: a = 6 | b = 4.5 | c = 7.5
Please note that the validation of a triangle could fail because of the fact
that double has precision failures during parsing regarding backed floating point !!!!
Entered values: p1: 6.000000 | p2: 4.500000 | p3: 7.500000
Your given triangle sites represent a 'square' triangle

Case 5: square triangle: a = 6_asasas | b = 4.5 | c = 7.5
Please note that the validation of a triangle could fail because of the fact
that double has precision failures during parsing regarding backed floating point !!!!
Entered values: p1: 6.000000 | p2: 4.500000 | p3: 7.500000
Your given triangle sites represent a 'square' triangle

Case 6: square triangle: a = 6 | b = 4.5_asasas | c = 7.5
Please note that the validation of a triangle could fail because of the fact
that double has precision failures during parsing regarding backed floating point !!!!
Entered values: p1: 6.000000 | p2: 4.500000 | p3: 7.500000
Your given triangle sites represent a 'square' triangle

Case 7: square triangle: a = 6 | b = 4.5 | c = 7.5_asasas
Please note that the validation of a triangle could fail because of the fact
that double has precision failures during parsing regarding backed floating point !!!!
Entered values: p1: 6.000000 | p2: 4.500000 | p3: 7.500000
Your given triangle sites represent a 'square' triangle
```

Diese Test zeigen das die Dreiecke korrekt erkannt werden sofern es keinen Ungenauigkeitsfehler bei den Seitenlängen (Dezimalwerte) gibt.

Hier sei auf die Fehlertoleranz von atoi hingewiesen, welche Argumente in Double Datentypen konvertiert, welche eigentlich nicht konvertierbar sein sollten.

3 Primfaktorenzerlegung

Der Folgende Teil der Dokuments dokumentiert die Aufgabenstellung Primfaktorenzerlegung, wobei eine ganze Zahl in Ihre Primfaktoren zerlegt werden soll.

3.1 Lösungsidee

Folgend ist die Lösungsidee zur Aufgabe Primfaktorenzerlegung angeführt.
Zuerst soll mittels Sieve of Eratosthense ein Array von Boolean aufgebaut werden, welches die Primzahlen mittels dem boolschen Wert am Index darstellt. Der Index mit boolschen Wert true ist also eine Primzahl. Dieses Array soll am Beginn des Programms initialisiert werden, sofern alle Vorbedingungen erfüllt sind. Die Primzahlen sollen bis 32565 ermittelt werden, obwohl theoretisch auch ein größerer Wertebereich abgedeckt werden kann, da für die Variablen der Datentyp Long Int verwendet werden soll.
Der Parameter, welcher beim Aufruf des Programms übergeben wird muss innerhalb der folgenden Grenzen sein: $1 < x < 32566$. (Obergrenze soll bis LONG_MAX erweitert werden können)

1. 0, 1 = Ungültige Primzahl
2. Überlauf, oder Segmentation fault, wenn Wertebereich von Long Int überschritten wird.

<u>Anschließend soll in einem Algorithmus die Primfaktorenfolge wie folgt ermittelt werden:</u>

1. Iteriere über die das Array mit den Primfaktoren
2. Wenn eine Primzahl gefunden wurde, versuche den Wert ohne Rest durch sie zu dividieren
    a. Wenn nein, gehe zu **1**
    b. Wenn ja, dann
        i. Setze Wert auf Divisionsergebnis
        ii. Merke ermittelte Primzahl
        iii. Bilde Produkt mit dieser Primzahl
        iv. Dekrementiere Index um 1, um neuen Wert erneut durch die Primzahl dividieren zu können
3. Führe diese Schritte aus solange
    a. Index kleiner als die Anzahl der Elemente im Array
    b. Index kleiner als der zu prüfende Wert
    c. Produkt der ermittelten Primzahlen ist kleiner gleich dem Wert
    d. Der Wert keine Primzahl ist
4. Prüfe ob zuletzt ermittelter Wert eine Primzahl ist
    a. Wenn ja, Merke diese Primzahl
    b. Wenn nein, dann konnte die Zahl nicht zerlegt werden (Zu wenig Primzahlen verfügbar)

Die ermittelten Primzahlen sollen dann mit folgenden Pattern ausgegeben werden.

E.g.: Prime factors of value: 256; 2^8

*Da diese Implementierung mit **aoti** arbeitet werden auch Zeichen konvertiert, die wie folgt aussehen. Arg = 6.6.6!"!§"sdfga wird zu 6, daher ist auf dies zu achten.*

Aufgrund der obigen Erkenntnis ist in Zukunft auf **atoi** zu verzichten, da es anscheinend sehr Fehlertolerant ist.

3.2 Implementierung

Der folgende Teil dieser Dokumentation behandelt den implementierten Source, sowie auch Test Source.

3.2.1 prime.h

```
/*
 * prime.h
 *
 *  Created on: Sep 20, 2014
 *      Author: cchet
 */
#include "common.h"

#ifndef PRIME_H_
#define PRIME_H

/* Defines the maximum of primes to be determined */
#define MAX_PRIM 32565

/**
 * Initializes an array of bool which index represents an prime
 * when the bool is set to true, otherwise it is not a prime.
 *
 * primeArray: the initializes bool array
 */
void initPrimes(bool*);

/**
 * Gets the prime factors of the given integer value.
 * The prime factors will be put in the given int[].
 *
 * calculatedPrimes: will contain the prime factors
 * primes: the bool[] containing the primes
 * value. the integer value to factorize
 * return count: the count of prime factors, -1 if the value could not be fully factorized.
 */
int GetPrimeFactors(long int* calculatedPrimes, bool* primes, long int value);

/**
 * Main function which gets called by the main program.
 * Checks for valid arguments and will return result
 *
 * argc: the argument count passed by the main program
 * argv: the arguments passed by the main program
 */
int ValidateNumber(int argc, char** argv);

/**
 * Same as ValidateNumber but aware of parsing errors, which atoi does not provide.
 * see: int ValidateNumber(int argc, char** argv)
 */
int ValidateNumberErrorAware(int argc, char** argv);
/**
 * Handles the result and prints the result message to the console.
```

```
 *
 * calculatedPrimes: contains the prime factors
 * count: the count of prime factors int he array
 * value: the value which got factorized.
 */
int HandleResult(long int* calculatedPrimes, int count, long int value);

#endif /* PRIME_H_ */
```

3.2.2 prime.c

```
/*
 * prime factorization.c
 *
 * This program is used for calculating the prime factors of an integer value.
 *
 *  Created on: Sep 18, 2014
 *      Author: cchet
 */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <errno.h>
#include <string.h>
#include "prime.h"

int ValidateNumber(int argc, char** argv) {
    bool primes[MAX_PRIM];
    long int calculatedPrimes[MAX_PRIM];
    int count;
    long int value;

    /* Check for invalid parameter count */
    if (argc != 3) {
        printf("Exactly one integer parameter must be given !!! count: %i \n",
                argc - 2);
        return EXIT_FAILURE;
    } /* if */

    /* Check for correct parameter type and value */
    if (((value = atoi(argv[2])) <= 1) || (value > MAX_PRIM)) {
        printf(
                "Parameter must be of type integer and must be within 2 - %i (inclusive) !!!
arg: %s\n",
                MAX_PRIM, argv[2]);
        return EXIT_FAILURE;
    } /* if */

    initPrimes(primes);
    count = GetPrimeFactors(calculatedPrimes, primes, value);
    return HandleResult(calculatedPrimes, count, value);
}

int ValidateNumberErrorAware(int argc, char** argv) {
    bool primes[MAX_PRIM];
    long int calculatedPrimes[MAX_PRIM];
    int count;
    long int value;
    char *tailptr = "";
    errno = -1;

    /* Check for invalid parameter count */
    if (argc != 3) {
        printf("Exactly one integer parameter must be given !!! count: %i \n",
                argc - 2);
        return EXIT_FAILURE;
    } /* if */

    /* Check for correct parameter type and value */
    value = strtol(argv[2], &tailptr, 10);
    if (errno == ERANGE) {
        printf("Given integer overflows range (long int)");
        return EXIT_FAILURE;
    } else if (*tailptr != '\0') {
        printf("Argument is no valid integer !!! arg: %s", argv[2]);
        return EXIT_FAILURE;
```

```
    } else if ((value <= 2) || (value > MAX_PRIM)) {
        printf(
                "Argument must be of type integer and must be within 2 - %i (inclusive) !!!
arg: %s\n",
                MAX_PRIM, argv[2]);
        return EXIT_FAILURE;
    } /* if */

    initPrimes(primes);
    count = GetPrimeFactors(calculatedPrimes, primes, value);
    return HandleResult(calculatedPrimes, count, value);
}

int GetPrimeFactors(long int* calculatedPrimes, bool* primes, long int value) {
    long int i;
    long int calculatedValue = 1;
    int count = 0;
    int calculatedIdx = 0;
    long int result = value;

    for (i = 2;
            (i < MAX_PRIM) && (calculatedValue < value) && (i <= result)
                    && (!primes[result]); i++) {
        if (primes[i]) {
            if ((result % i) == 0) {
                result = result / i;
                calculatedValue *= i;
                calculatedPrimes[calculatedIdx] = i;
                count++;
                calculatedIdx++;
                i--;
            } /*if*/
        } /* if */
    } /* for */

    /* Could split value into its prime factors */
    if (primes[result]) {
        calculatedPrimes[calculatedIdx] = result;
        count++;
    }
    /* prime factor split failed */
    else {
        count = -1;
    } /* if */

    return count;
}

int HandleResult(long int* calculatedPrimes, int count, long int value) {
    long int i = 0;
    long int j = 0;
    int pot = 1;

    if (count != -1) {
        printf("Prime factors of value: %li; ", value);
        while (i < count) {
            pot = 1;
            j = (i + 1);
            while ((j < count) && (calculatedPrimes[i] == calculatedPrimes[j])) {
                pot++;
                j++;
            } /* while */
            printf("%li^%i", calculatedPrimes[i], pot);
            i += pot;
            if (i < count) {
                printf(" * ");
            } else {
                printf("\n");
            } /* if */
        } /* while */
        return EXIT_SUCCESS;
    } else {
        printf("Value: %li could not be factorized\n", value);
        return EXIT_FAILURE;
    } /* if */
}
```

```
void initPrimes(bool* primes) {
    long int i, j;
    primes[0] = false;
    primes[1] = false;
    for (i = 2; i <= MAX_PRIM; i++) {
        primes[i] = true;
    } /* for */
    for (i = 2; i <= MAX_PRIM; i++) {
        if (primes[i]) {
            for (j = 2 * i; j <= MAX_PRIM; j += i) {
                primes[j] = false;
            } /* for */
        } /*if*/
    } /* for */
}
```

3.2.3 prime_test.sh

```
#!/bin/sh
## Varaibles used during build and tests ##
APPLICATION_TYPE=2
SOURCE_MAIN=../main.c
BINARY_MAIN=bin/main_prime_test.bin
SOURCE_TRIANGLE=../source/triangle.c
SOURCE_PRIME=../source/prime.c
INCLUDE_HEADERS_COMMONS=../../Commons/header
INCLUDE_HEADERS=../header/

echo "################################################################################"
echo "These are the tests for the prime_factorizer"
echo "################################################################################ \n"

## Compile the source file ##
## build triangle source
echo "compiling source file '$SOURCE_MAIN' and dependencies to target $BINARY_MAIN \n"
echo "$ gcc -g -I$INCLUDE_HEADERS_COMMONS -I$INCLUDE_HEADERS $SOURCE_MAIN $SOURCE_TRIANGLE
$SOURCE PRIME -o $BINARY MAIN -lm\n"
gcc -g -I$INCLUDE_HEADERS_COMMONS -I$INCLUDE_HEADERS $SOURCE_MAIN $SOURCE_TRIANGLE
$SOURCE_PRIME -o $BINARY_MAIN -lm

## Check if file has been compiled
if [ -f "$BINARY_MAIN" ]
then
    echo "source compiled starting tests now ...\n"
else
    echo "compile of $SOURCE_MAIN failed\n"
    exit 1
fi

################################################################
## Test wrong parameter count                                 ##
################################################################
echo "--------------------------------------------------------------------------------"
echo "Testing invalid parameter count"
echo "--------------------------------------------------------------------------------"
## Case 1
echo "Case 1: no param"
$BINARY_MAIN $APPLICATION_TYPE
echo "\n"

## Case 2
echo "Case 2: 2 param"
$BINARY_MAIN $APPLICATION_TYPE 123 456
echo "\n"

## Case 3
echo "Case 3: 3 param"
$BINARY_MAIN $APPLICATION_TYPE 123 456 789
echo "\n"

################################################################
## Test invalid parameter range                               ##
################################################################
echo "--------------------------------------------------------------------------------"
echo "Testing with invalid parameter range  "
```

```
echo "------------------------------------------------------------------------------"
## Case 1
echo "Case 1: 0"
$BINARY_MAIN $APPLICATION_TYPE 0
echo "\n"

## Case 2
echo "Case 2: 1"
$BINARY_MAIN $APPLICATION_TYPE 1
echo "\n"

## Case 3
echo "Case 3: 32566 out fo range"
$BINARY_MAIN $APPLICATION_TYPE 32566
echo "\n"

## Case 4
echo "Case 4: -1 "
$BINARY_MAIN $APPLICATION_TYPE -1
echo "\n"

################################################################
## Test invalid parameters                                    ##
################################################################
echo "------------------------------------------------------------------------------"
echo "Testing with character parameter  "
echo "------------------------------------------------------------------------------"
## Case 1
echo "Case 1: abc"
$BINARY_MAIN $APPLICATION_TYPE abc
echo "\n"

## Case 2
echo "Case 2: asdfadsfdf 454544"
$BINARY_MAIN $APPLICATION_TYPE asdfadsfdf_454544
echo "\n"

################################################################
## Test valid parameters                                      ##
################################################################
echo "------------------------------------------------------------------------------"
echo "Testing with valid params"
echo "------------------------------------------------------------------------------"
## Case 1
echo "Case 1: 10"
$BINARY_MAIN $APPLICATION_TYPE 10
echo "\n"

## Case 2
echo "Case 2: 256"
$BINARY_MAIN $APPLICATION_TYPE 256
echo "\n"

## Case 3
echo "Case 3: 6534"
$BINARY_MAIN $APPLICATION_TYPE 6534
echo "\n"

## Case 4
echo "Case 4: 13332"
$BINARY_MAIN $APPLICATION_TYPE 13332
echo "\n"

## Case 5
echo "Case 5: 167 is already prime"
$BINARY_MAIN $APPLICATION_TYPE 167
echo "\n"

## Case 6
echo "Case 6: 3673 is already prime"
$BINARY_MAIN $APPLICATION_TYPE 3673
echo "\n"

## Case 7
echo "Case 7: 6.8 reduced to int part "
$BINARY_MAIN $APPLICATION_TYPE 6.8
```

```
echo "\n"

## Case 8
echo "Case 8: 6.6.6!"!ﻙ"sdfga "
$BINARY_MAIN $APPLICATION_TYPE 6.6.6!"!ﻙ"sdfga
echo "\n"

## Case 9
echo "Case 9: 6458_aadfdfdfsadfsadfsdf "
$BINARY_MAIN $APPLICATION_TYPE 6458_aadfdfdfsadfsadfsdf
echo "\n"
echo "tests done !!!!!"

## Cleanup binary
echo "Remove binaries ..."
rm -rf $BINARY_MAIN
echo "Binaries removed"
exit 0
```

3.3 Tests

Der folgende Teil dieser Dokumentation behandelt die Testfälle für dieses C-Programm.

Die Tests wurden im oben angeführten Shell Skript implementiert und müssen wie im Punkt **1. Allgemein** beschrieben ausgeführt werden.

3.3.1 Testfälle Anzahl Parameter

Diese Tests testen die Fehlerbehandlung von einer falschen Parameteranzahl.

```
--------------------------------------------------------------------------------
Testing invalid parameter count
--------------------------------------------------------------------------------
Case 1: no param
Exactly one integer parameter must be given !!! count: 0


Case 2: 2 param
Exactly one integer parameter must be given !!! count: 2


Case 3: 3 param
Exactly one integer parameter must be given !!! count: 3
```

Es muss exakt ein Argument übergeben werden, jede Abweichung davon verursacht einen Fehler, der dem Aufrufer mitgeteilt wird.

3.3.2 Testfälle ungültige Parameter

Diese Tests testen die Fehlerbehandlung von ungültigen Parametern.

```
--------------------------------------------------------------------------------
Testing with character parameter
--------------------------------------------------------------------------------
Case 1: abc
Parameter must be of type integer and must be within 2 - 32565 (inclusive) !!! arg: abc


Case 2: asdfadsfdf_454544
Parameter must be of type integer and must be within 2 - 32565 (inclusive) !!! arg: asdfadsfdf_454544
```

In diesem Fall ist zu sehen, dass ungültige Argumente nur durch einen ungültigen Character am Anfang des Strings erkannt werden und daher 0 zurückgeliefert wird, was als ungültige Eingabe erkannt wird. Bei den Funktionstests ohne Fehler ist zu erkennen, dass wenn die numerischen Werte am Anfang des Strings stehen würden, dann würde atoi keine Fehler produzieren und solange dieser numerische Wert über 2 ist, wird das Programm ohne Fehler ausgeführt.

3.3.3 Testfälle ungültige Parameter Wertebereich

Diese Tests testen die Fehlerbehandlung von ungültigen Parametern Wertebereichen.

```
--------------------------------------------------------------------------------
Testing with invalid parameter range
--------------------------------------------------------------------------------
Case 1: 0
Parameter must be of type integer and must be within 2 - 32565 (inclusive) !!! arg: 0


Case 2: 1
Parameter must be of type integer and must be within 2 - 32565 (inclusive) !!! arg: 1


Case 3: 32566 out fo range
Parameter must be of type integer and must be within 2 - 32565 (inclusive) !!! arg: 32566


Case 4: -1
Parameter must be of type integer and must be within 2 - 32565 (inclusive) !!! arg: -1
```

Der Wertebereich der gültigen Integer wird vom Programm vorgegeben und ist mindestens 2 und maximal INT_MAX belegt und kann höchstens auf LONG_MAX erweitert werden, da der Datentyp mit Long Int gewählt wurde.

3.3.4 Testfälle ohne Fehler

Diese Tests testen das Programm ohne fehlerhafte Eingaben.

```
--------------------------------------------------------------------------------
Testing with valid params
--------------------------------------------------------------------------------
Case 1: 10
Prime factors of value: 10; 2^1 * 5^1


Case 2: 256
Prime factors of value: 256; 2^8


Case 3: 6534
Prime factors of value: 6534; 2^1 * 3^3 * 11^2


Case 4: 13332
Prime factors of value: 13332; 2^2 * 3^1 * 11^1 * 101^1


Case 5: 167 is already prime
Prime factors of value: 167; 167^1


Case 6: 3673 is already prime
Prime factors of value: 3673; 3673^1


Case 7: 6.8 reduced to int part
Prime factors of value: 6; 2^1 * 3^1


Case 8: 6.6.6!!Â§sdfga
Prime factors of value: 6; 2^1 * 3^1


Case 9: 6458_aadfdfdfsadfsadfsdf
Prime factors of value: 6458; 2^1 * 3229^1
```

Hier ist zu sehen, das aoti nicht unbedingt geeignet für ein Programm ist, da es sehr Fehlertolerant ist. Es scheint als müsste nur am Anfang des Strings ein numerischer Wert definiert werden und schon konvertiert atoi diesen und ignoriert die fehlerhaften Strings Teile einfach. Daher ist atoi in Zukunft nicht zu verwenden.

4 Primfaktorenzerlegung mit ERRNO

Dieser Teil der Dokumentation dokumentiert die Primfaktorenzerlegung mit ERRNO.
Bei ERRNO handelt es sich um eine globale Variable, die von Funktionen gesetzt wird damit Fehler nach außen bekannt gemacht werden können.

4.1 Lösungsidee

Ich habe mich für die Funktion **strtol** entschieden, die einen String in einen Long Int Datentyp konvertiert.
Es kann hierbei ein ein Char Pointer mitübergeben werden, welcher mit dem Teil des Strings gesetzt wird, welcher nicht in ein Long Int konvertiert werden kann. Über diesen Teilstring kann nach dem konvertieren ermittelt werden, ob der gesamte String konvertiert hat werden können, oder nur Teile davon.
Da diese Funktion bei einem Konvertierungsfehler den Rückgabewert entweder auf LONG_MAX oder LONG_MIN setzt, ist diese Art der Fehlerbehandlung besser da man sich nicht auf LONG_MAX und LONG_INT alleine verlassen kann, da diese Werte durchaus auch gewollt sein können.
Man könnte hierbei nicht unterscheiden ob es sich um einen Konvertierungsfehler oder um eine Eingabe dieses Wertes handelt.

Damit der Aufwand so gering wie möglich gehalten wird, soll das bestehende Programm herangezogen und um eine Funktion erweitert werden die als Einstiegspunkt in dieses Unterprogram dient. In dieser Funktion soll die Fehlerbehandlung erfolgen, der Rest des Programms soll wiederverwendet werden.

4.2 Implementierung

Folgend werden nur die Modifikationen des Programms gelistet, da sich sonst nichts weiter verändert hat.

4.2.1 C – Source

Die Funktion, die eingeführt wurde um die Fehlerbehandlung zu realisieren. Sie ist ebenfalls auch im obigen Soruce von prime.c enthalten.

```
int ValidateNumberErrorAware(int argc, char** argv) {
    bool primes[MAX_PRIM];
    long int calculatedPrimes[MAX_PRIM];
    int count;
    long int value;
    char *tailptr = "";
    errno = -1;

    /* Check for invalid parameter count */
    if (argc != 3) {
        printf("Exactly one integer parameter must be given !!! count: %i \n",
                argc - 2);
        return EXIT_FAILURE;
    } /* if */

    /* Check for correct parameter type and value */
    value = strtol(argv[2], &tailptr, 10);
    if (errno == ERANGE) {
        printf("Given integer overflows range (long int)");
        return EXIT_FAILURE;
    } else if (*tailptr != '\0') {
        printf("Argument is no valid integer !!! arg: %s", argv[2]);
        return EXIT_FAILURE;
    } else if ((value <= 2) || (value > MAX_PRIM)) {
        printf(
                "Argument must be of type integer and must be within 2 - %i (inclusive) !!!
arg: %s\n",
                MAX_PRIM, argv[2]);
        return EXIT_FAILURE;
    } /* if */

    initPrimes(primes);
    count = GetPrimeFactors(calculatedPrimes, primes, value);
    return HandleResult(calculatedPrimes, count, value);
}
```

4.2.2 Shell Script – Tests

Folgend ist das Shell Skript angeführt über welches das Programm getestet wird. Wobei es sich hier um dieselben Test handelt wie ohne Fehlerbehandlung, bis auf die neu hinzugefügten Tests, die die Fehlerbehandlung testen.

```
#!/bin/sh
## Varaibles used during build and tests ##
APPLICATION_TYPE=3
SOURCE_MAIN=../main.c
BINARY_MAIN=bin/main_prime_errno_test.bin
SOURCE_TRIANGLE=../source/triangle.c
SOURCE_PRIME=../source/prime.c
INCLUDE_HEADERS_COMMONS=../../Commons/header
INCLUDE_HEADERS=../header/

echo "################################################################################"
echo "These are the tests for the prime_factorizer which is error aware"
echo "################################################################################ \n"

## Compile the source file ##
## build triangle source
echo "compiling source file '$SOURCE_MAIN' and dependencies to target $BINARY_MAIN \n"
```

```
echo "$ gcc -g -I$INCLUDE_HEADERS_COMMONS -I$INCLUDE_HEADERS $SOURCE_MAIN $SOURCE_TRIANGLE
$SOURCE_PRIME -o $BINARY_MAIN -lm\n"
gcc -g -I$INCLUDE_HEADERS_COMMONS -I$INCLUDE_HEADERS $SOURCE_MAIN $SOURCE_TRIANGLE
$SOURCE_PRIME -o $BINARY_MAIN -lm

## Check if file has been compiled
if [ -f "$BINARY_MAIN" ]
then
    echo "source compiled starting tests now ...\n"
else
    echo "compile of $SOURCE_MAIN failed\n"
    exit 1
fi

################################################################
## Test wrong parameter count                                 ##
################################################################
echo "-------------------------------------------------------------------------------"
echo "Testing invalid parameter count"
echo "-------------------------------------------------------------------------------"
## Case 1
echo "Case 1: no param"
$BINARY_MAIN $APPLICATION_TYPE
echo "\n"

## Case 2
echo "Case 2: 2 param"
$BINARY_MAIN $APPLICATION_TYPE 123 456
echo "\n"

## Case 3
echo "Case 3: 3 param"
$BINARY_MAIN $APPLICATION_TYPE 123 456 789
echo "\n"

################################################################
## Test invalid parameter range                               ##
################################################################
echo "-------------------------------------------------------------------------------"
echo "Testing with invalid parameter range  "
echo "-------------------------------------------------------------------------------"
## Case 1
echo "Case 1: 0"
$BINARY_MAIN $APPLICATION_TYPE 0
echo "\n"

## Case 2
echo "Case 2: 1"
$BINARY_MAIN $APPLICATION_TYPE 1
echo "\n"

## Case 3
echo "Case 3: 32566 out fo range"
$BINARY_MAIN $APPLICATION_TYPE 32566
echo "\n"

## Case 4
echo "Case 4: -1 "
$BINARY_MAIN $APPLICATION_TYPE -1
echo "\n"

################################################################
## Test invalid parameters                                    ##
################################################################
echo "-------------------------------------------------------------------------------"
echo "Testing with character parameter  "
echo "-------------------------------------------------------------------------------"
## Case 1
echo "Case 1: abc"
$BINARY_MAIN $APPLICATION_TYPE abc
echo "\n"

## Case 2
echo "Case 2: asdfadsfdf_454544"
$BINARY_MAIN $APPLICATION_TYPE asdfadsfdf_454544
echo "\n"
```

```
## Case 3
echo "Case 3: 6.8 reduced to int part "
$BINARY_MAIN $APPLICATION_TYPE 6.8
echo "\n"

## Case 4
echo "Case 4: 6.6.6!"!ยง"sdfga "
$BINARY_MAIN $APPLICATION_TYPE 6.6.6!"!ยง"sdfga
echo "\n"

## Case 5
echo "Case 5: 6458_aadfdfdfsadfsadfsdf "
$BINARY_MAIN $APPLICATION_TYPE 6458_aadfdfdfsadfsadfsdf
echo "\n"

#################################################################
## Test valid parameters                                       ##
#################################################################
echo "-------------------------------------------------------------------------------"
echo "Testing with valid params"
echo "-------------------------------------------------------------------------------"
## Case 1
echo "Case 1: 10"
$BINARY_MAIN $APPLICATION_TYPE 10
echo "\n"

## Case 2
echo "Case 2: 256"
$BINARY_MAIN $APPLICATION_TYPE 256
echo "\n"

## Case 3
echo "Case 3: 6534"
$BINARY_MAIN $APPLICATION_TYPE 6534
echo "\n"

## Case 4
echo "Case 4: 13332"
$BINARY_MAIN $APPLICATION_TYPE 13332
echo "\n"

## Case 5
echo "Case 5: 167 is already prime"
$BINARY_MAIN $APPLICATION_TYPE 167
echo "\n"

## Case 6
echo "Case 6: 3673 is already prime"
$BINARY_MAIN $APPLICATION_TYPE 3673
echo "\n"
echo "tests done !!!!!"

## Cleanup binary
echo "Remove binaries ..."
rm -rf $BINARY_MAIN
echo "Binaries removed"
exit 0
```

4.3 Tests

Der folgende Teil dieser Dokumentation behandelt die Testfälle für dieses C-Programm.

Die Tests wurden im oben angeführten Shell Skript implementiert und müssen wie im Punkt **1. Allgemein** beschrieben ausgeführt werden.

4.3.1 Testfälle Anzahl Parameter

Diese Tests testen die Fehlerbehandlung von einer falschen Parameteranzahl.

```
--------------------------------------------------------------------------------
Testing invalid parameter count
--------------------------------------------------------------------------------
Case 1: no param
Exactly one integer parameter must be given !!! count: 0


Case 2: 2 param
Exactly one integer parameter must be given !!! count: 2


Case 3: 3 param
Exactly one integer parameter must be given !!! count: 3
```

Es muss exakt ein Argument übergeben werden, jede Abweichung davon verursacht einen Fehler, der dem Aufrufer mitgeteilt wird.

4.3.2 Testfälle ungültige Parameter

Diese Tests testen die Fehlerbehandlung von ungültigen Parametern.

```
--------------------------------------------------------------------------------
Testing with character parameter
--------------------------------------------------------------------------------
Case 1: abc
Argument is no valid integer !!! arg: abc

Case 2: asdfadsfdf_454544
Argument is no valid integer !!! arg: asdfadsfdf_454544

Case 3: 6.8 reduced to int part
Argument is no valid integer !!! arg: 6.8

Case 4: 6.6.6!!Â§sdfga
Argument is no valid integer !!! arg: 6.6.6!!Â§sdfga

Case 5: 6458_aadfdfdfsadfsadfsdf
Argument is no valid integer !!! arg: 6458_aadfdfdfsadfsadfsdf
```

Hier ist zu sehen, das die Argumente, welche bei Verwendung von atoi gültig waren, es jetzt nicht mehr sind. Daher ist strtol aoti vorzuziehen.

4.3.3 Testfälle ungültige Parameter Wertebereich

Diese Tests testen die Fehlerbehandlung von ungültigen Parametern Wertebereichen.

```
--------------------------------------------------------------------------------
Testing with invalid parameter range
--------------------------------------------------------------------------------
Case 1: 0
Argument must be of type integer and must be within 2 - 32565 (inclusive) !!! arg: 0


Case 2: 1
Argument must be of type integer and must be within 2 - 32565 (inclusive) !!! arg: 1


Case 3: 32566 out fo range
Argument must be of type integer and must be within 2 - 32565 (inclusive) !!! arg: 32566


Case 4: -1
Argument must be of type integer and must be within 2 - 32565 (inclusive) !!! arg: -1
```

Der Wertebereich der gültigen Integer wird vom Programm vorgegeben und ist mindestens 2 und maximal INT_MAX belegt und kann höchstens auf LONG_MAX erweitert werden, da der Datentyp mit Long Int gewählt wurde.

4.3.4 Testfälle ohne Fehler

Diese Tests testen das Programm ohne fehlerhafte Eingaben.

```
--------------------------------------------------------------------------------
Testing with valid params
--------------------------------------------------------------------------------
Case 1: 10
Prime factors of value: 10; 2^1 * 5^1


Case 2: 256
Prime factors of value: 256; 2^8


Case 3: 6534
Prime factors of value: 6534; 2^1 * 3^3 * 11^2


Case 4: 13332
Prime factors of value: 13332; 2^2 * 3^1 * 11^1 * 101^1


Case 5: 167 is already prime
Prime factors of value: 167; 167^1


Case 6: 3673 is already prime
Prime factors of value: 3673; 3673^1
```

Diese Test zeigen, dass nur noch gültige Integer Argumente verwendet werden können, da strtol nicht fehlertolerant ist.