Track_hw1

Generated by Doxygen 1.8.17

Chapter 1

File Index

1.1 File List

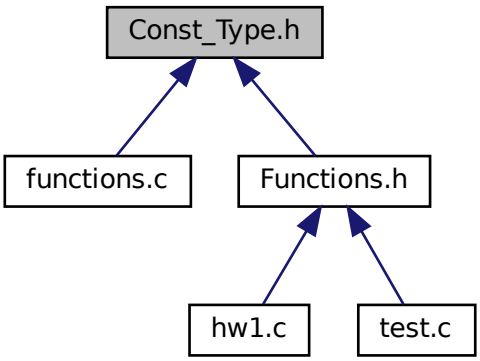
Here is a list of all files with brief descriptions:

Chapter 2

File Documentation

2.1 Const_Type.h File Reference

This graph shows which files directly or indirectly include this file:



Typedefs

- typedef enum Bool Bool

Enumerations

- enum Bool { FALSE, TRUE }

Variables

- const int SS_INF_ROOTS

2.1.1 Typedef Documentation

2.1.1.1 Bool

```
typedef enum Bool Bool
```

2.1.2 Enumeration Type Documentation

2.1.2.1 Bool

```
enum Bool
```

Enumerator

FALSE	
TRUE	

2.1.3 Variable Documentation

2.1.3.1 SS_INF_ROOTS

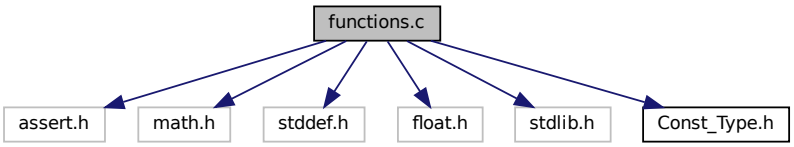
```
const int SS_INF_ROOTS
```

2.2 functions.c File Reference

```
#include <assert.h>
#include <math.h>
#include <stddef.h>
#include <float.h>
#include <stdlib.h>
```

```
#include "Const_Type.h"
```

Include dependency graph for functions.c:



Functions

- double checkDoubleOverflow (double d1, double d2, int x)

  Check for correctness of the result of multiplying two double numbers.
- Bool EqualZero (double d)

  Checks that a floating point number is zero.
- Bool EqualDouble (double d1, double d2)

  Checks that floating point numbers are equal.
- void LinSolve (double a, double b, double ∗x1)

  Solve a linear equation ax = b.
- void nCheckDouble (double a, double b, double c, double ∗x1, double ∗x2)

  Check the correctness of the entered data.
- int SquareSolve (double a, double b, double c, double ∗x1, double ∗x2)

  Solve a quadratic equation ax^2 + bx + c = 0.
- void Sort (double ∗x1, double ∗x2)

  Sorts two variables. Puts a larger number in the first.

Variables

- const int SS_INF_ROOTS = -1

2.2.1 Function Documentation

2.2.1.1 checkDoubleOverflow()

```
double checkDoubleOverflow (
            double d1,
            double d2,
            int x )
```

Check for correctness of the result of multiplying two double numbers.

Parameters

in	*d1*	is the first number.
in	*d2*	second number.
in	*x*	(optional) the number by which to multiply the result of the multiplication.

Returns

res result of multiplication: res = x ∗ d1 ∗ d2.

2.2.1.2 EqualDouble()

```
Bool EqualDouble (
            double d1,
            double d2 )
```

Checks that floating point numbers are equal.

Parameters

in	*d1*	Number to check.
in	*d2*	The number to check.

Returns

True / False if number is / is not equal to zero.

2.2.1.3 EqualZero()

```
Bool EqualZero (
            double d )
```

Checks that a floating point number is zero.

Parameters

in	*d*	The number to check.

Returns

True / False if number is / is not equal to zero.

2.2.1.4 LinSolve()

```
void LinSolve (
            double a,
            double b,
            double * x1 )
```

Solve a linear equation ax = b.

Parameters

in	*a*	coefficient a.
in	*b*	coefficient b.
out	*x1*	Pointer to the first root.

Returns

Void

Note

check for a! = 0 is enabled. When testing, the function sets

2.2.1.5 nCheckDouble()

```
void nCheckDouble (
            double a,
            double b,
            double c,
            double * x1,
            double * x2 )
```

Check the correctness of the entered data.

Parameters

in	*a*	coefficient a.
in	*b*	coefficient b.
in	*c*	coefficient c.
out	*x1*	Pointer to the first root.
out	*x2*	Pointer to the second root.

Returns

void function

Note

Does not check data type out of bounds.

2.2.1.6 Sort()

```
void Sort (
            double * x1,
            double * x2 )
```

Sorts two variables. Puts a larger number in the first.

Parameters

in	*x1*	- pointer to the first variable
in	*x2*	- pointer to the first variable
out	*x1*	- large variable
out	*x2*	- smaller variable

Returns

void

2.2.1.7 SquareSolve()

```
int SquareSolve (
            double a,
            double b,
            double c,
            double * x1,
            double * x2 )
```

Solve a quadratic equation ax ^ 2 + bx + c = 0.

Parameters

in	*a*	coefficient a.
in	*b*	coefficient b.
in	*c*	coefficient c.
out	*x1*	Pointer to the first root.
out	*x2*	Pointer to the second root.

Returns

Number of roots of the equation

Note

in cases where the equation has no roots returns SS_INF_ROOTS

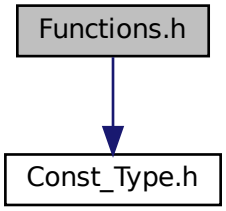
2.2.2 Variable Documentation

2.2.2.1 SS_INF_ROOTS

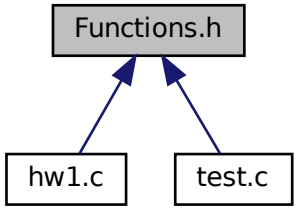
```
const int SS_INF_ROOTS = -1
```

2.3 Functions.h File Reference

```
#include "Const_Type.h"
```
Include dependency graph for Functions.h:



This graph shows which files directly or indirectly include this file:



Functions

- double checkDoubleOverflow (double d1, double d2, int x)

  Check for correctness of the result of multiplying two double numbers.
- void nCheckDouble (double a, double b, double c, double ∗x1, double ∗x2)

  Check the correctness of the entered data.
- Bool EqualZero (double d)

  Checks that a floating point number is zero.
- Bool EqualDouble (double d1, double d2)

  Checks that floating point numbers are equal.

- void LinSolve (double a, double b, double ∗x1)

  Solve a linear equation ax = b.
- int SquareSolve (double a, double b, double c, double ∗x1, double ∗x2)

  Solve a quadratic equation ax^ 2 + bx + c = 0.
- void Sort (double ∗x1, double ∗x2)

  Sorts two variables. Puts a larger number in the first.

2.3.1 Function Documentation

2.3.1.1 checkDoubleOverflow()

```
double checkDoubleOverflow (
            double d1,
            double d2,
            int x )
```

Check for correctness of the result of multiplying two double numbers.

Parameters

in	*d1*	is the first number.
in	*d2*	second number.
in	*x*	(optional) the number by which to multiply the result of the multiplication.

Returns

res result of multiplication: res = x ∗ d1 ∗ d2.

2.3.1.2 EqualDouble()

```
Bool EqualDouble (
            double d1,
            double d2 )
```

Checks that floating point numbers are equal.

Parameters

in	*d1*	Number to check.
in	*d2*	The number to check.

Returns

True / False if number is / is not equal to zero.

2.3.1.3 EqualZero()

```
Bool EqualZero (
            double d )
```

Checks that a floating point number is zero.

Parameters

in	*d*	The number to check.

Returns

True / False if number is / is not equal to zero.

2.3.1.4 LinSolve()

```
void LinSolve (
            double a,
            double b,
            double * x1 )
```

Solve a linear equation ax = b.

Parameters

in	*a*	coefficient a.
in	*b*	coefficient b.
out	*x1*	Pointer to the first root.

Returns

Void

Note

check for a! = 0 is enabled. When testing, the function sets

2.3.1.5 nCheckDouble()

```
void nCheckDouble (
            double a,
            double b,
            double c,
            double * x1,
            double * x2 )
```

Check the correctness of the entered data.

Parameters

in	*a*	coefficient a.
in	*b*	coefficient b.
in	*c*	coefficient c.
out	*x1*	Pointer to the first root.
out	*x2*	Pointer to the second root.

Returns

void function

Note

Does not check data type out of bounds.

2.3.1.6 Sort()

```
void Sort (
            double * x1,
            double * x2 )
```

Sorts two variables. Puts a larger number in the first.

Parameters

in	*x1*	- pointer to the first variable
in	*x2*	- pointer to the first variable
out	*x1*	- large variable
out	*x2*	- smaller variable

Returns

void

2.3.1.7 SquareSolve()

```
int SquareSolve (
            double a,
            double b,
            double c,
            double * x1,
            double * x2 )
```

Solve a quadratic equation ax ^ 2 + bx + c = 0.

Parameters

in	*a*	coefficient a.
in	*b*	coefficient b.
in	*c*	coefficient c.
out	*x1*	Pointer to the first root.
out	*x2*	Pointer to the second root.

Returns

Number of roots of the equation

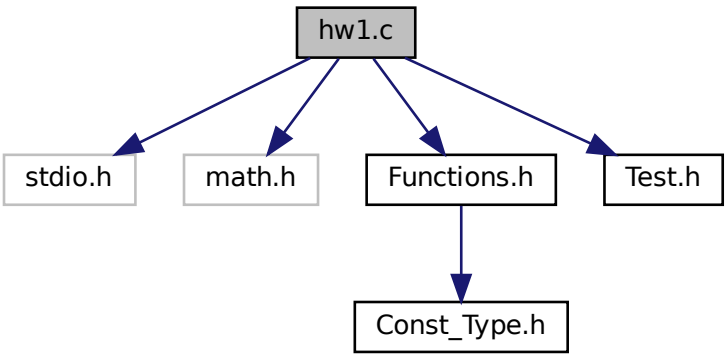
Note

in cases where the equation has no roots returns SS_INF_ROOTS

2.4 hw1.c File Reference

```
#include <stdio.h>
#include <math.h>
#include "Functions.h"
#include "Test.h"
```

Include dependency graph for hw1.c:



Functions

- int main ()

2.4.1 Function Documentation

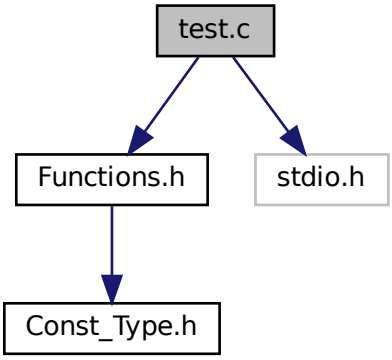
2.4.1.1 main()

```
int main ( )
```

2.5 test.c File Reference

```
#include "Functions.h"
#include <stdio.h>
```

Include dependency graph for test.c:



Functions

- void Unit_LinSolve (double a, double b, double correct_value, int line)

  Tests the LinSolve function - solution ax + b = 0.
- void Unit_SquareSolve_roots_exist (double a, double b, double c, double correct_root1, double correct_root2, int correct_n, int line)

  Tests the SquareSolve function - solution ax ^ 2 + bx + c = 0 if there is at least one root.
- void Unit_SquareSolve_roots_no_exist (double a, double b, double c, int correct_n, int line)

  Tests the SquareSolve function - the solution ax ^ 2 + bx + c = 0, if there are no roots or it is any number.
- void Unit_EqualZero (double d, Bool correct_equal, int line)
- void test_LinSolve ()

  Tests the LinSolve function - solution ax + b = 0.
- void test_SquareSolve ()

  Tests the SquareSolve function - solution ax ^ 2 + bx + c = 0.
- void Test_EqualZero ()

2.5.1 Function Documentation

2.5.1.1 Test_EqualZero()

```
void Test_EqualZero ( )
```

2.5.1.2 test_LinSolve()

```
void test_LinSolve ( )
```

Tests the LinSolve function - solution ax + b = 0.

Returns

Void

Note

Calls Unit_LinSolve multiple times.

2.5.1.3 test_SquareSolve()

```
void test_SquareSolve ( )
```

Tests the SquareSolve function - solution ax ^ 2 + bx + c = 0.

Returns

void

Note

The function calls Unit_SquareSolve several times

2.5.1.4 Unit_EqualZero()

```
void Unit_EqualZero (
            double d,
            Bool correct_equal,
            int line )
```

2.5.1.5 Unit_LinSolve()

```
void Unit_LinSolve (
            double a,
            double b,
            double correct_value,
            int line )
```

Tests the LinSolve function - solution ax + b = 0.

Parameters

in	*a*	coefficient a.
in	*b*	coefficient b.
in	*correct_value*	is the correct function value.
in	*line*	- line number with error.

Returns

Void

Note

Validation checks are made before, a! = 0 whenever the function is called.

2.5.1.6 Unit_SquareSolve_roots_exist()

```
void Unit_SquareSolve_roots_exist (
            double a,
            double b,
            double c,
            double correct_root1,
            double correct_root2,
            int correct_n,
            int line )
```

Tests the SquareSolve function - solution ax ^ 2 + bx + c = 0 if there is at least one root.

Parameters

in	*a*	coefficient a.
in	*b*	coefficient b.
in	*c*	coefficient c.
in	*correct_root1*	is the first correct root.
in	*correct_root2*	second correct root.
in	*correct_n*	number of roots.
in	*line*	- line number with error.

Returns

Void

Note

The function passed the validation against the data entered earlier. If the roots are the same, then it is necessary to put in the two variables are the same root.

2.5.1.7 Unit_SquareSolve_roots_no_exist()

```
void Unit_SquareSolve_roots_no_exist (
            double a,
            double b,
            double c,
            int correct_n,
            int line )
```

Tests the SquareSolve function - the solution ax ^ 2 + bx + c = 0, if there are no roots or it is any number.

Parameters

in	*a*	coefficient a.
in	*b*	coefficient b.
in	*c*	coefficient c.
in	*correct_n*	number of roots.
in	*line*	- line number with error.

Returns

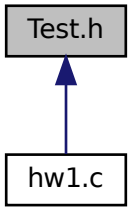
Void

Note

The function passed the validation against the data entered earlier. If the roots are the same, then it is necessary to put in the two variables are the same root.

2.6 Test.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void test_LinSolve ()

  Tests the LinSolve function - solution ax + b = 0.
- void test_SquareSolve ()

  Tests the SquareSolve function - solution ax ^ 2 + bx + c = 0.
- void Test_EqualZero ()

2.6.1 Function Documentation

2.6.1.1 Test_EqualZero()

```
void Test_EqualZero ( )
```

2.6.1.2 test_LinSolve()

```
void test_LinSolve ( )
```

Tests the LinSolve function - solution ax + b = 0.

Returns

Void

Note

Calls Unit_LinSolve multiple times.

2.6.1.3 test_SquareSolve()

```
void test_SquareSolve ( )
```

Tests the SquareSolve function - solution ax ^ 2 + bx + c = 0.

Returns

void

Note

The function calls Unit_SquareSolve several times