

IVS_PROJEKT_2_KALKULACKA

Generated by Doxygen 1.10.0

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

backend/ operation.c	??
backend/ operation.h	??
backend/ operation_tests.c	??
frontend/ main.c	??

Chapter 2

File Documentation

2.1 backend/operation.c File Reference

```
#include <mpfr.h>
#include <math.h>
#include <malloc.h>
#include <string.h>
#include "stdlib.h"
```

Macros

- #define `MAX_PRECISION_DEC` 30
- #define `PRECISION_DECIMALS` 100
- #define `PRECISION_BITS` $\text{ceil}(\text{PRECISION_DECIMALS} * \log_2(10))$
- #define `rounding` MPFR_RNDZ

Functions

- void `remove_substring` (char *str, const char *sub)
- void `removeTrailingZeros` (char *str)
- char * `convertToString` (mpfr_t number)
- char * `op_add` (char *addend1, char *addend2)
- char * `op_sub` (char *minuend, char *subtrahend)
subtraction for two numbers
- char * `op_mul` (char *multiplicand, char *multiplier)
multiplication for two numbers
- char * `op_div` (char *dividend, char *divisor)
division for two numbers
- char * `op_factorial` (char *factor)
math factorial
- char * `op_pow` (char *base, char *exponent)
- char * `op_root` (char *radicand, char *index)
general root
- char * `op_sin` (char *argument)
general root

2.1.1 Macro Definition Documentation

2.1.1.1 MAX_PRECISION_DEC

```
#define MAX_PRECISION_DEC 30
```

2.1.1.2 PRECISION_BITS

```
#define PRECISION_BITS ceil(PRECISION_DECIMALS * log2(10))
```

2.1.1.3 PRECISION_DECIMALS

```
#define PRECISION_DECIMALS 100
```

2.1.1.4 rounding

```
#define rounding MPFR_RNDZ
```

2.1.2 Function Documentation

2.1.2.1 convertToString()

```
char * convertToString (  
    mpfr_t number )
```

2.1.2.2 op_add()

```
char * op_add (  
    char * addend1,  
    char * addend2 )
```

Parameters

<i>addend1</i>	First number string for addition
<i>addend2</i>	Second number string for addition

Returns

result Resulting number as string

2.1.2.3 op_div()

```
char * op_div (  
    char * dividend,  
    char * divisor )
```

division for two numbers

Parameters

<i>dividend</i>	Number that will be divided by the
<i>divisor</i>	
<i>divisor</i>	Number that will divide
<i>dividend</i>	

Returns

result Quotient as string

2.1.2.4 op_factorial()

```
char * op_factorial (
    char * factor )
```

math factorial

Parameters

<i>factor</i>	will determine to where we shall multiply numbers for example factor of five will result in 120
---------------	---

Returns

result resulting factorial of

Parameters

<i>factor</i>	
---------------	--

2.1.2.5 op_mul()

```
char * op_mul (
    char * multiplicand,
    char * multiplier )
```

multiplication for two numbers

Parameters

<i>multiplicand</i>	Number that will be multiplied by the
<i>multiplier</i>	
<i>multiplier</i>	Number that will multiply the
<i>multiplicand</i>	

Returns

result Product of the multiplicand and multiplier

2.1.2.6 op_pow()

```
char * op_pow (
    char * base,
    char * exponent )
```

Parameters

<i>exponent</i>	power of
<i>base</i>	
<i>base</i>	will be exponentiated by
<i>exponent</i>	
<i>exponent</i>	will be used to exponentiate
<i>base</i>	

Returns

result is the exponentiated

Parameters

<i>base</i>	by
<i>exponent</i>	

2.1.2.7 op_root()

```
char * op_root (
    char * radicand,
    char * index )
```

general root

Parameters

<i>radicand</i>	will be exponentiated by
<i>index</i>	
<i>index</i>	will be used to exponentiate
<i>base</i>	

Returns

result from root with index of

Parameters

<i>index</i>	from
<i>radicand</i>	

2.1.2.8 op_sin()

```
char * op_sin (
    char * argument )
```

general root

Parameters

<i>argument</i>	variable in radians used for sin
-----------------	----------------------------------

Returns

result resulting number from sin(

Parameters

<i>argument</i>)	
-------------------	--

2.1.2.9 op_sub()

```
char * op_sub (
    char * minuend,
    char * subtrahend )
```

subtraction for two numbers

Parameters

<i>minuend</i>	Number that is going to be subtracted from
<i>subtrahend</i>	Number that will be used to subtract from minuend

Returns

result Product of the minuend and subtrahend

2.1.2.10 remove_substring()

```
void remove_substring (
    char * str,
    const char * sub )
```

2.1.2.11 removeTrailingZeros()

```
void removeTrailingZeros (
    char * str )
```

2.2 backend/operation.h File Reference

Functions

- char * [op_add](#) (char *addend1, char *addend2)
- char * [op_sub](#) (char *minuend, char *subtrahend)
subtraction for two numbers
- char * [op_mul](#) (char *multiplicand, char *multiplier)
multiplication for two numbers
- char * [op_div](#) (char *dividend, char *divisor)
division for two numbers
- char * [op_factorial](#) (char *factor)
math factorial
- char * [op_pow](#) (char *base, char *exponent)
- char * [op_root](#) (char *radicand, char *index)
general root
- char * [op_sin](#) (char *argument)
general root

2.2.1 Function Documentation

2.2.1.1 op_add()

```
char * op_add (
    char * addend1,
    char * addend2 )
```

Parameters

<i>addend1</i>	First number string for addition
<i>addend2</i>	Second number string for addition

Returns

result Resulting number as string

2.2.1.2 op_div()

```
char * op_div (
    char * dividend,
    char * divisor )
```

division for two numbers

Parameters

<i>dividend</i>	Number that will be divided by the
<i>divisor</i>	
<i>divisor</i>	Number that will divide
<i>dividend</i>	

Returns

result Quotient as string

2.2.1.3 op_factorial()

```
char * op_factorial (
    char * factor )
```

math factorial

Parameters

<i>factor</i>	will determine to where we shall multiply numbers for example factor of five will result in 120
---------------	---

Returns

result resulting factorial of

Parameters

<i>factor</i>	
---------------	--

2.2.1.4 op_mul()

```
char * op_mul (
    char * multiplicand,
    char * multiplier )
```

multiplication for two numbers

Parameters

<i>multiplicand</i>	Number that will be multiplied by the
<i>multiplier</i>	
<i>multiplier</i>	Number that will multiply the
<i>multiplicand</i>	

Returns

result Product of the multiplicand and multiplier

2.2.1.5 op_pow()

```
char * op_pow (
    char * base,
    char * exponent )
```

Parameters

<i>exponent</i>	power of
<i>base</i>	
<i>base</i>	will be exponentiated by
<i>exponent</i>	
<i>exponent</i>	will be used to exponentiate
<i>base</i>	

Returns

result is the exponentiated

Parameters

<i>base</i>	by
<i>exponent</i>	

2.2.1.6 op_root()

```
char * op_root (
    char * radicand,
    char * index )
```

general root

Parameters

<i>radicand</i>	will be exponentiated by
<i>index</i>	
<i>index</i>	will be used to exponentiate
<i>base</i>	

Returns

result from root with index of

Parameters

<i>index</i>	from
<i>radicand</i>	

2.2.1.7 op_sin()

```
char * op_sin (
    char * argument )
```

general root

Parameters

<i>argument</i>	variable in radians used for sin
-----------------	----------------------------------

Returns

result resulting number from sin(

Parameters

<i>argument</i>)	
-------------------	--

2.2.1.8 op_sub()

```
char * op_sub (
    char * minuend,
    char * subtrahend )
```

subtraction for two numbers

Parameters

<i>minuend</i>	Number that is going to be subtracted from
<i>subtrahend</i>	Number that will be used to subtract from minuend

Returns

result Product of the minuend and subtrahend

2.3 operation.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
```

```
00003 char *op_add(char *addend1, char *addend2);
00004 char *op_sub(char *minuend, char *subtrahend);
00005 char *op_mul(char *multiplicand, char *multiplier);
00006 char *op_div(char *dividend, char *divisor);
00007 char *op_factorial(char *factor);
00008 char *op_pow(char *base, char *exponent);
00009 char *op_root(char *radicand, char *index);
00010 char *op_sin(char *argument);
```

2.4 backend/operation_tests.c File Reference

```
#include "operation.h"
#include "string.h"
#include <assert.h>
```

Functions

- void [test_addition](#) (char *number1, char *number2, char *expected)
- void [test_subtraction](#) (char *number1, char *number2, char *expected)
- void [test_multiplication](#) (char *number1, char *number2, char *expected)
- void [test_division](#) (char *number1, char *divisor, char *expected)
- void [test_factorial](#) (char *number, char *expected)
- void [test_power](#) (char *base, char *exp, char *expected)
- void [test_root](#) (char *number, char *exponent, char *expected)
- void [test_sin](#) (char *number, char *expected)
- int [main](#) (void)

2.4.1 Function Documentation

2.4.1.1 main()

```
int main (
    void )
```

2.4.1.2 test_addition()

```
void test_addition (
    char * number1,
    char * number2,
    char * expected )
```

2.4.1.3 test_division()

```
void test_division (
    char * number1,
    char * divisor,
    char * expected )
```

2.4.1.4 test_factorial()

```
void test_factorial (
    char * number,
    char * expected )
```

2.4.1.5 test_multiplication()

```
void test_multiplication (
    char * number1,
    char * number2,
    char * expected )
```

2.4.1.6 test_power()

```
void test_power (
    char * base,
    char * exp,
    char * expected )
```

2.4.1.7 test_root()

```
void test_root (
    char * number,
    char * exponent,
    char * expected )
```

2.4.1.8 test_sin()

```
void test_sin (
    char * number,
    char * expected )
```

2.4.1.9 test_subtraction()

```
void test_subtraction (
    char * number1,
    char * number2,
    char * expected )
```

2.5 frontend/main.c File Reference

```
#include "raylib.h"
#include "raygui.h"
#include "style_jungle.h"
#include "operation.h"
```

Macros

- #define `RAYGUI_IMPLEMENTATION`

Enumerations

- enum `operation` {
 `plus` = 1 , `minus` , `mult` , `division` ,
 `sinus` , `fact` , `root` , `power` }

Functions

- void `addNumberToCurrNum` (char *currNum, char *number)
 helper adds number to currNum string and handles edge cases
- int `main` ()

Variables

- short `errState` = 0

2.5.1 Macro Definition Documentation

2.5.1.1 RAYGUI_IMPLEMENTATION

```
#define RAYGUI_IMPLEMENTATION
```

2.5.2 Enumeration Type Documentation

2.5.2.1 operation

```
enum operation
```

Enumerator

plus	
minus	
mult	
division	
sinus	
fact	
root	
power	

2.5.3 Function Documentation

2.5.3.1 addNumberToCurrNum()

```
void addNumberToCurrNum (  
    char * currNum,  
    char * number )
```

helper adds number to currNum string and handles edge cases

Parameters

<i>currNum</i>	array that hold current number you're inputing
<i>number</i>	number you want to input

Returns

void

2.5.3.2 main()

```
int main (  
    void )
```

2.5.4 Variable Documentation

2.5.4.1 errState

```
short errState = 0
```

