SmartCalc

Description of the project

The implementation of an extended version of a calculator.
In addition to basic arithmetic operations such as add/subtract and multiply/divide, this calculator has the ability to calculate arithmetic expressions by following the order, as well as some mathematical functions (sine, cosine, logarithm, etc.).
Besides calculating expressions, it supports the usage of the x variable and the graphing of the corresponding function.
Also it has a credit and deposit calculator's modes.

This project was developed with:

1. C language
2. GUI implementation based on Qt
3. Plotting graphs via QCustomPlot
4. Makefile
5. Doxygen

Author

naginisa

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

C **credit**	Structure for working with input data of annuity credit
C **credit_calc**	Structure for working with output data of annuity credit
C **deposit_info**	Structure for working with input and output data of deposit
C **Month**	Structure for working with deposits and withdrawals in the deposit settlement
C **Stack**	Definition of a stack structure based on a singly linked list

Here is a list of all documented file members with links to the documentation:

- a -

- add_number() : **stack.h**
- annuity_calculations() : **extra_calcs.h**

- b -

- binary_operation() : **stack.h**
- braces() : **data_validation.h**

- c -

- calc_result() : **stack.h**
- calc_stack : **stack.h**
- calculations() : **stack.h**
- check_braces() : **data_validation.h**
- check_comma_with_dot() : **data_validation.h**
- check_dots() : **data_validation.h**
- check_funcs() : **data_validation.h**
- check_mod() : **data_validation.h**
- check_number() : **data_validation.h**
- check_operators() : **data_validation.h**
- check_spaces() : **data_validation.h**
- check_unar() : **data_validation.h**
- check_x() : **data_validation.h**
- credit_input : **extra_calcs.h**
- credit_output : **extra_calcs.h**
- credit_type : **extra_calcs.h**

- d -

- deposit_calculation() : **extra_calcs.h**
- deposit_info : **extra_calcs.h**
- destroy_stack() : **stack.h**

- i -

- is_func() : **data_validation.h**
- is_funcs() : **stack.h**
- is_mod() : **data_validation.h**
- is_number() : **data_validation.h**
- is_operator() : **data_validation.h**

- m -

- math_precendence() : **stack.h**
- Month : **extra_calcs.h**

- n -

- new_polish_stack() : **stack.h**

- p -

- pop() : **stack.h**
- push() : **stack.h**
- push_operators_and_functions() : **stack.h**

- r -

- reverse_stack() : **stack.h**

- s -

- string_validation() : **data_validation.h**

- t -

- type_t : **stack.h**

- u -

- unary_operation() : **stack.h**

data_validation.h File Reference

Header file describing the functions used to validate a string. More...

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Go to the source code of this file.

Functions

int **string_validation** (char *input, char *string)
Validates a mathematical expression.

void **check_spaces** (char *string, char *copy)
Removes spaces between characters.

int **check_comma_with_dot** (char *string)
Performs an initial check for the correct placement of dots in the expression and changes the ',' symbol to the '.' symbol.

int **check_braces** (char *string)
Checks the correctness of parentheses in an expression and their pairing.

int **is_number** (char symb)
Checks whether a character is a number.

int **is_operator** (char symb)
Checks whether a character is an operation sign.

int **check_dots** (char *string, int float_dot, int i)
Checks the correctness of the dot notation.

int **check_number** (char *string, int *i, int *float_dot)
Checks the correctness of the number notation.

int **check_funcs** (char *string, int *i)
Checks the correctness of the function name.

int **is_func** (char *string, int i)
Checks whether the symbol is an operator sign.

int **check_operators** (char *string, int i)
Checks the correctness of the operation signs.

int **is_mod** (char *string, int i)
Checks for the presence of the "mod" operation in the given expression string.

int **check_mod** (char *string, int i)
Checks the correctness of the "mod" operation.

int **check_x** (char *string, int i)
Проверяет корректность записи переменной 'x' в строке

int **braces** (char *string, int i)
Performs an additional check for the correct use of parentheses in the string.

void **check_unar** (char *string, int i)
Checks if the symbol '-' is an unary minus, replaces the symbol with '~' if an unary minus is encountered.

Detailed Description

Header file describing the functions used to validate a string.

Function Documentation

braces()

```
int braces ( char *  string,
             int     i
           )
```

Performs an additional check for the correct use of parentheses in the string.

Parameters

string string-expression

i index of the current symbol in the string

Returns

returns 1 in case of an error, 0 - if there is no error

check_braces()

```
int check_braces ( char *  string )
```

Checks the correctness of parentheses in an expression and their pairing.

Parameters

string string-expression

Returns

returns 1 if an error occurs, 0 if the expression is successfully read

check_comma_with_dot()

int check_comma_with_dot (char * string)

Performs an initial check for the correct placement of dots in the expression and changes the ',' symbol to the '.' symbol.

Parameters

string string-expression

Returns

returns 1 if an error occurred, 0 if the expression was successfully read

◆ check_dots()

```
int check_dots ( char * string,
                 int    float_dot,
                 int    i
               )
```

Checks the correctness of the dot notation.

Parameters

string string-expression

float_dot Flag indicating the presence of a decimal point in the expression (1 - if the point has been encountered before, 0 - if there was no point)

i Index of the character being checked

Returns

returns 1 in case of an error, 0 - if there is no error

◆ check_funcs()

```
int check_funcs ( char *  string,
                  int *   i
                )
```

Checks the correctness of the function name.

Parameters

string string-expression

i pointer to the current character in the string

Returns

returns 1 in case of an error, 0 - if there is no error

◆ check_mod()

```
int check_mod ( char *  string,
                int     i
              )
```

Checks the correctness of the "mod" operation.

Parameters

string string-expression

i index of the current symbol in the string

Returns

returns 1 in case of an error, 0 - if there is no error

◆ check_number()

```
int check_number ( char *  string,
                   int *   i,
                   int *   float_dot
                 )
```

Checks the correctness of the number notation.

Parameters

string string-expression

i pointer to the current character in the string

float_dot pointer to the value of the flag indicating the presence of a decimal point in the expression (1 - if the point has been encountered before, 0 - if there was no point)

Returns

returns 1 in case of an error, 0 - if there is no error

◆ check_operators()

```
int check_operators ( char *  string,
                      int     i
                    )
```

Checks the correctness of the operation signs.

Parameters

string string-expression

i index of the current symbol in the string

Returns

returns 1 in case of an error, 0 - if there is no error.

◆ check_spaces()

```
void check_spaces ( char *  string,
                    char *  copy
                  )
```

Removes spaces between characters.

Parameters

string source string

copy copy of the source string without spaces

◆ check_unar()

```
void check_unar ( char *  string,
                  int     i
                )
```

Checks if the symbol '-' is an unary minus, replaces the symbol with '~' if an unary minus is encountered.

Parameters

string string-expression

i index of the current symbol in the string

◆ check_x()

```
int check_x ( char *  string,
              int     i
            )
```

Проверяет корректность записи переменной 'x' в строке

Parameters

string string-expression

i index of the current symbol in the string

Returns

returns 1 in case of an error, 0 - if there is no error

◆ is_func()

```
int is_func ( char *  string,
              int     i
            )
```

Checks whether the symbol is an operator sign.

Parameters

string string-expression

i index of the current symbol in the string

Returns

κeturns 0 if the check is successful, 1 if it is not an operator

◆ is_mod()

```
int is_mod ( char *  string,
             int     i
           )
```

Checks for the presence of the "mod" operation in the given expression string.

Parameters

string string-expression

i index of the current symbol in the string

Returns

returns 1 if the operation is present, 0 - if it is not

◆ is_number()

```
int is_number ( char  symb )
```

Checks whether a character is a number.

Parameters

symb character to be checked

Returns

returns 1 if the character is a number, 0 if the character is not a number

◆ is_operator()

int is_operator (char symb)

Checks whether a character is an operation sign.

Parameters

symb character to be checked

Returns

returns 1 if the symbol is an operation sign, 0 if the symbol is not an operation

◆ string_validation()

int string_validation (char * input,
char * string
)

Validates a mathematical expression.

Parameters

input string-expression received from the user

string converted copy of the expression string after validation

Returns

returns 1 if an error occurred, 0 if the expression was successfully read

stack.h File Reference

Header file with a description of functions and structures used for working with a stack. More...

```
#include "../back/data_validation.h"
```

Go to the source code of this file.

Classes

struct **Stack**
Definition of a stack structure based on a singly linked list. More...

Typedefs

typedef struct **Stack** **calc_stack**
Definition of a stack structure based on a singly linked list.

Enumerations

enum **type_t** {
Number = 0 , **X** = 1 , **Plus** = 2 , **Minus** = 3 ,
Multiply = 4 , **Division** = 5 , **Pow** = 6 , **Mod** = 7 ,
Cos = 8 , **Sin** = 9 , **Tan** = 10 , **Acos** = 11 ,
Asin = 12 , **Atan** = 13 , **Sqrt** = 14 , **Ln** = 15 ,
Log = 16 , **Lbrace** = 17 , **Rbrace** = 18 , **UnarMinus** = 19
}
Definition of type_t data type using enum enumeration to simplify stack operations and define data types for parsing in RPN.

Functions

void **push** (**calc_stack** **stack, double value, int priority, **type_t** type)
Adding a new node to the stack.

void **pop** (**calc_stack** **stack)
Removing a node from the stack.

void **destroy_stack** (**calc_stack** **stack)
Complete clearing of the stack from all elements.

calc_stack * **reverse_stack** (**calc_stack** **stack)
Reverse of the stack.

int **add_number** (char *string, char *num)
Writes number characters to an array.

double **calc_result** (**calc_stack** *rpn)
Performs expression calculation.

double **binary_operation** (double a, double b, int data_type)

	Calculates operation with two values.
double	**unary_operation** (double a, int type) Calculates operation with one value.
double	**calculations** (char *string, double x) Performs a call to other functions to obtain the expression result.
int	**push_operators_and_functions** (char *string, **calc_stack** **stack) Determines the type of operation and adds it to the stack structure.
int	**is_funcs** (char symb) Determines whether the symbol is a function name.
int	**math_precendence** (char symb) Determines the priority of operation execution.
calc_stack *	**new_polish_stack** (char *string, double x) Performs parsing of infix expression lexemes and adds elements to the stack, converting the expression to postfix using the shunting-yard algorithm.

Detailed Description

Header file with a description of functions and structures used for working with a stack.

Typedef Documentation

◆ calc_stack

typedef struct **Stack calc_stack**

Definition of a stack structure based on a singly linked list.

Parameters

value	the value of the number
priority	the priority value of the mathematical operation
data_type	the data type placed in the structure
next	pointer to the next node in the stack

Function Documentation

◆ add_number()

```
int add_number ( char *  string,
                 char *  num
               )
```

Writes number characters to an array.

Parameters

string string-expression

num character array into which number characters are written, including decimal point

Returns

number of characters that form the number (its length)

◆ binary_operation()

```
double binary_operation ( double  a,
                          double  b,
                          int     data_type
                        )
```

Calculates operation with two values.

Parameters

a value of first operand

b value of second operand

data_type type of operation

Returns

result of the calculation

◆ calc_result()

```
double calc_result ( calc_stack *  rpn )
```

Performs expression calculation.

Parameters

rpn stack of operands and operations in reverse Polish notation

Returns

result of the calculation

◆ calculations()

```
double calculations ( char *  string,
                      double  x
                    )
```

Performs a call to other functions to obtain the expression result.

Parameters

string string-expression

x value of variable x (0 if not specified)

Returns

result of the calculation

◆ destroy_stack()

```
void destroy_stack ( calc_stack **  stack )
```

Complete clearing of the stack from all elements.

Parameters

stack pointer to a pointer to the stack

◆ is_funcs()

```
int is_funcs ( char  symb )
```

Determines whether the symbol is a function name.

Parameters

symb symbol to check for matching

Returns

returns 1 if the symbol belongs to the function name, 0 if the condition is not met

◆ math_precendence()

int math_precendence (char symb)

Determines the priority of operation execution.

Parameters

symb symbol to check for matching

Returns

returns 1 for addition/subtraction, 2 – multiplication/division/modulo, 3 – power, 4 – functions, 5 – unary minus

◆ new_polish_stack()

calc_stack * new_polish_stack (char * string,
double x
)

Performs parsing of infix expression lexemes and adds elements to the stack, converting the expression to postfix using the shunting-yard algorithm.

Parameters

string string-expression

x value of variable x (0 if not specified)

Returns

stack with elements in reverse Polish notation

◆ pop()

void pop (**calc_stack** ** stack)

Removing a node from the stack.

Parameters

stack pointer to a pointer to the stack

◆ push()

```
void push ( calc_stack **  stack,
            double         value,
            int            priority,
            type_t         type
          )
```

Adding a new node to the stack.

Parameters

stack pointer to a pointer to the stack where the new element should be placed

value value of the element, if it is a number (0 if it is not)

priority priority value of the operation (0 if it is a number)

type data type being placed in the structure

◆ push_operators_and_functions()

```
int push_operators_and_functions ( char *          string,
                                   calc_stack **   stack
                                 )
```

Determines the type of operation and adds it to the stack structure.

Parameters

string pointer to character in the string

stack stack to which the element is added

Returns

length of the operation characters

◆ reverse_stack()

```
calc_stack * reverse_stack ( calc_stack **  stack )
```

Reverse of the stack.

Parameters

stack pointer to pointer to the stack to be reversed

Returns

new reversed stack, original is deleted

◆ unary_operation()

```
double unary_operation ( double  a,
                         int     type
                       )
```

Calculates operation with one value.

Parameters

a value of the operand

type type of operation

Returns

result of the calculation

extra_calcs.h File Reference

Header file with function descriptions used for working with credit and deposit calculators. More...

```
#include <math.h>
```

Go to the source code of this file.

Classes

struct	**credit** Structure for working with input data of annuity credit. More...
struct	**credit_calc** Structure for working with output data of annuity credit. More...
struct	**deposit_info** Structure for working with input and output data of deposit. More...
struct	**Month** Structure for working with deposits and withdrawals in the deposit settlement. More...

Typedefs

typedef struct **credit**	**credit_input** Structure for working with input data of annuity credit.
typedef struct **credit_calc**	**credit_output** Structure for working with output data of annuity credit.
typedef struct **deposit_info**	**deposit_info** Structure for working with input and output data of deposit.
typedef struct **Month**	**Month** Structure for working with deposits and withdrawals in the deposit settlement.

Enumerations

enum	**credit_type** { **annuity** = 1 , **differentiated** = 2 } enumeration for determining the type of credit More...

Functions

credit_output	**annuity_calculations** (**credit_input** data) Calculates annuity credit.
int	**deposit_calculation** (struct **deposit_info** *dep, **Month** *months) Calculates deposit.

Detailed Description

Header file with function descriptions used for working with credit and deposit calculators.

Typedef Documentation

credit_input

typedef struct **credit credit_input**

Structure for working with input data of annuity credit.

Parameters

amount total loan amount

term loan term

percent interest rate

credit_output

typedef struct **credit_calc credit_output**

Structure for working with output data of annuity credit.

Parameters

interest_overpayment overpayment on the loan

total_payments total payments on the loan

monthly_payment monthly payment

deposit_info

typedef struct **deposit_info deposit_info**

Structure for working with input and output data of deposit.

Parameters

input_summ	deposit amount
temp_of_placement	placement term
percent	interest rate
tax_rate	tax rate
frequency_of_payments	payment frequency
capitalization_of_interest	interest capitalization
list_of_deposits	list of deposits of funds
list_of_partial_withdrawals	list of partial withdrawals
accrued_interest	accrued interest
result_summ	amount on the deposit at the end of the term
tax_amount	tax amount for deduction

◆ Month

typedef struct **Month Month**

Structure for working with deposits and withdrawals in the deposit settlement.

Parameters

plus_value	deposit amount
minus_value	withdrawal amount

Enumeration Type Documentation

◆ credit_type

enum **credit_type**

enumeration for determining the type of credit

Parameters

annuity	1 - annuity
differentiated	2 - differentiated

Function Documentation

annuity_calculations()

credit_output annuity_calculations (**credit_input** data)

Calculates annuity credit.

Parameters

data data structure with input data for calculation

Returns

returns a structure containing data on overpayment on the loan, monthly payment, and total payments on the loan

deposit_calculation()

int deposit_calculation (struct **deposit_info** * dep,
Month * months
)

Calculates deposit.

Parameters

dep pointer to the **deposit_info** structure containing information about the deposit

months pointer to an array of **Month** structures representing months

Returns

returns 1 on successful calculation, 0 in case of error, 3 - impossible operations with zero