**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**федеральное ГОСУДАРСТВЕННОЕ бюджетное ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

**Кафедра программного обеспечения вычислительной техники и автоматизированных систем**

**Курсовая работа**

**Разработка и реализация библиотеки для работы с числом decimal на языке программирования СИ**

Выполнил:

Студент группы: ВТ – 231

Шевченко Д. А.

Принял:

Лукьянов А.М.

Белгород

2024

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| Введение..................................................................................................................... | 3 |
| Постановка задачи..................................................................................................... | 3 |
| Описание решения задачи......................................................................................... | 4 |
| Техническая информация.......................................................................................... | 4 |
| Основная часть........................................................................................................... | 5 |
| Вывод……………………………………………………………………………….. | 16 |

**Введение**

В рамках современного программирования существует множество задач, связанных с обработкой числовых данных высокой точности. Одной из таких задач является работа с вещественными числами, где требуется высокая точность вычислений без потери значащих цифр. Стандартный тип данных float или double, используемый в большинстве языков программирования, имеет ограниченную точность и может приводить к ошибкам округления при выполнении сложных математических операций. Для решения этой проблемы часто применяется специальный тип данных – десятичное число (decimal), который позволяет хранить дробные значения с фиксированной точностью.

**ПОСТАНОВКА ЗАДАЧИ**

Разработать библиотеку decimal на языке программирования Си.

**ОПИСАНИЕ РЕШЕНИЯ ЗАДАЧИ**

Для реализации матричной библиотеки на языке программирования Си были созданы несколько файлов самих функций на языке программирования Си, заголовочный h файл и файл с тестовыми сценариями, который находится в папке tests. В этих файлах содержится вся необходимая логика для работы с decimal, а также тесты для проверки корректности выполнения операций.

**ТЕХНИЧЕСКАЯ ИНФОРМАЦИЯ**

Проект направлен на создание инструмента, который позволит разработчикам легко интегрировать работу с десятичными числами в свои приложения, обеспечивая высокую точность вычислений и минимизируя ошибки округления. Библиотека будет разработана с учетом требований к производительности, удобству использования и совместимости с существующими стандартами языка программирования.

**ОСНОВНАЯ ЧАСТЬ**

**Импорт библиотек и создание заголовочного файла**

Проект начинается с подключения стандартных библиотек языка Си . Затем определяются коды ошибок с помощью перечисления и максимальные, а так же минимальные значения нашего decimal.

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define SIZE\_S21\_DECIMAL 4

#define HIGHEST\_BIT\_IN\_32\_BIT\_PART 31

#define COUNT\_BITS\_IN\_PART 32

#define START\_SCALE\_BIT 16

#define LAST\_SCALE\_BIT 23

#define MAX\_HIGHT\_BIT 95

#define MAX\_POWER 28

#define BUFF\_SIZE 128

#define s21\_abs(x) ((x) < 0) ? (-(x)) : ((x))

#define DECIMAL\_MAX \

(s21\_decimal) { \

{ 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0 } \

}

#define DECIMAL\_MIN \

(s21\_decimal) { \

{ 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 1 << HIGHEST\_BIT\_IN\_32\_BIT\_PART } \

}

**Создание структур**

Этот фрагмент кода определяет структуру s21\_decimal для хранения десятичных чисел в виде массива битов, а также перечисления ARITHMETHIC\_RES для отслеживания результатов арифметических операций (успешного выполнения, выхода за пределы допустимых значений или попытки деления на ноль), RESULT для общего статуса операции (успех или ошибка) и BITS для обозначения различных частей числа (младшей, средней, старшей и шкалы).

typedef struct {

int bits[SIZE\_S21\_DECIMAL]; // Массив для хранения битов числа типа s21\_decimal

} s21\_decimal;

typedef enum {

ARITHMETHIC\_OK, // Операция выполнена успешно

ARITHMETHIC\_BIG\_INFINITY, // Число вышло за пределы допустимого диапазона (слишком большое)

ARITHMETHIC\_SMALL\_INFINITY, // Число вышло за пределы допустимого диапазона (слишком маленькое)

ARITHMETHIC\_ZERO\_DIV // Попытка деления на ноль

} ARITHMETHIC\_RES;

typedef enum {

OK, // Операция выполнена без ошибок

ERROR // Произошла ошибка при выполнении операции

} RESULT;

typedef enum { LOW\_BIT, MID\_BIT, HIGH\_BIT, SCALE } BITS;

**Функции**

После определения структуры matrix\_t и необходимых макросов, я перехожу к объявлению функций для работы с матрицами.

// Функция сложения двух чисел типа s21\_decimal

extern int s21\_add(s21\_decimal value\_1, s21\_decimal value\_2, s21\_decimal \*result);

// Функция вычитания одного числа типа s21\_decimal из другого

extern int s21\_sub(s21\_decimal value\_1, s21\_decimal value\_2, s21\_decimal \*result);

// Функция умножения двух чисел типа s21\_decimal

extern int s21\_mul(s21\_decimal value\_1, s21\_decimal value\_2, s21\_decimal \*result);

// Функция деления одного числа типа s21\_decimal на другое

extern int s21\_div(s21\_decimal value\_1, s21\_decimal value\_2, s21\_decimal \*result);

/\*Функция сравнения двух чисел типа s21\_decimal. Возвращает TRUE, если первое число меньше второго, иначе возвращает FALSE.\*/

extern int s21\_is\_less(s21\_decimal num1, s21\_decimal num2);

/\*Функция сравнения двух чисел типа s21\_decimal. Возвращает TRUE, если первое число меньше или равно второму, иначе возвращает FALSE.\*/

extern int s21\_is\_less\_or\_equal(s21\_decimal num1, s21\_decimal num2);

/\*Функция сравнения двух чисел типа s21\_decimal. Возвращает TRUE, если первое число больше второго, иначе возвращает FALSE.\*/

extern int s21\_is\_greater(s21\_decimal num1, s21\_decimal num2);

/\*Функция сравнения двух чисел типа s21\_decimal. Возвращает TRUE, если первое число больше или равно второму, иначе возвращает FALSE.\*/

extern int s21\_is\_greater\_or\_equal(s21\_decimal num1, s21\_decimal num2);

/\*Функция сравнения двух чисел типа s21\_decimal. Возвращает TRUE, если оба числа равны, иначе возвращает FALSE.\*/

extern int s21\_is\_equal(s21\_decimal num1, s21\_decimal num2);

/\*Функция сравнения двух чисел типа s21\_decimal. Возвращает TRUE, если числа не равны, иначе возвращает FALSE.\*/

extern int s21\_is\_not\_equal(s21\_decimal num1, s21\_decimal num2);

/\*Конвертирует целое число в число типа s21\_decimal. Возвращает 0 при успешном преобразовании, 1 - в случае ошибки.\*/

extern int s21\_from\_int\_to\_decimal(int src, s21\_decimal \*dst);

/\*Конвертирует число с плавающей точкой в число типа s21\_decimal. Возвращает 0 при успешном преобразовании, 1 - в случае ошибки.\*/

extern int s21\_from\_float\_to\_decimal(float src, s21\_decimal \*dst);

/\*Конвертирует число типа s21\_decimal в целое число. Возвращает 0 при успешном преобразовании, 1 - в случае ошибки.\*/

extern int s21\_from\_decimal\_to\_int(s21\_decimal src, int \*dst);

/\*Конвертирует число типа s21\_decimal в число с плавающей точкой. Возвращает 0 при успешном преобразовании, 1 - в случае ошибки.\*/

extern int s21\_from\_decimal\_to\_float(s21\_decimal src, float \*dst);

/\*Округляет указанное Decimal число до ближайшего целого числа в сторону отрицательной бесконечности.

\* Возвращаемое значение - код ошибки: 0 - OK || 1 - ошибка вычисления

\*/

extern int s21\_floor(s21\_decimal value, s21\_decimal \*result);

/\*Округляет Decimal до ближайшего целого числа.

\* Возвращаемое значение - код ошибки: 0 - OK || 1 - ошибка вычисления

\*/

extern int s21\_round(s21\_decimal value, s21\_decimal \*result);

/\*Возвращает целые цифры указанного Decimal числа; любые дробные цифры отбрасываются, включая конечные нули.

\* Возвращаемое значение - код ошибки: 0 - OK || 1 - ошибка вычисления

\*/

extern int s21\_truncate(s21\_decimal value, s21\_decimal \*result);

/\*Возвращает результат умножения указанного Decimal на -1.

\* Возвращаемое значение - код ошибки: 0 - OK || 1 - ошибка вычисления

\*/

extern int s21\_negate(s21\_decimal value, s21\_decimal \*result);

// ------------------------------------

// Bit Manipulation Functions

// ------------------------------------

/\*Получает знак числа типа s21\_decimal. Возвращает 0, если число положительное, и 1, если число отрицательное.\*/

int Get\_Sign(s21\_decimal number);

/\*Устанавливает знак числа типа s21\_decimal. Если параметр sign равен 0, то число становится положительным, если 1 - отрицательным.\*/

void Set\_Sign(s21\_decimal \*number, int sign);

/\*Получает масштаб числа типа s21\_decimal. Масштаб определяет количество знаков после запятой.\*/

int Get\_Scale(s21\_decimal number);

/\*Устанавливает масштаб числа типа s21\_decimal. Параметр scale определяет новое значение масштаба.\*/

void Set\_Scale(s21\_decimal \*number, int scale);

/\*Получает бит с указанным индексом в числе типа s21\_decimal. Индекс может варьироваться от 0 до 127.\*/

int Get\_Bit(s21\_decimal number, int bit);

/\*Устанавливает бит с указанным индексом в числе типа s21\_decimal. Значение бита определяется параметром sign (0 или 1).\*/

void Set\_Bit(s21\_decimal \*number, int bit, int sign);

/\*Сдвигает все биты числа типа s21\_decimal влево на одну позицию. Возвращает 0, если операция прошла успешно, иначе 1.\*/

int Shift\_Left(s21\_decimal \*number);

/\*Сдвигает все биты числа типа s21\_decimal вправо на одну позицию. Возвращает 0, если операция прошла успешно, иначе 1.\*/

int Shift\_Right(s21\_decimal \*number);

// ------------------------------------

// Initialization Functions

// ------------------------------------

/\*Инициализирует структуру типа s21\_decimal нулями. Используется для создания нового пустого числа.\*/

void Init\_Decimal\_Num(s21\_decimal \*number);

// ------------------------------------

// Simple Arithmetic Operations

// ------------------------------------

// Складывает два числа типа s21\_decimal.

int Simple\_Add(s21\_decimal value1, s21\_decimal value2, s21\_decimal \*result);

// Вычитает одно число типа s21\_decimal из другого.

void Simple\_Sub(s21\_decimal value1, s21\_decimal value2, s21\_decimal \*result);

// Перемножает два числа типа s21\_decimal.

int Simple\_Mul(s21\_decimal value1, s21\_decimal value2, s21\_decimal \*result);

// Делит одно число типа s21\_decimal на другое.

s21\_decimal Simple\_Div(s21\_decimal value1, s21\_decimal value2, s21\_decimal \*result);

// ------------------------------------

// Операции сравнения

// ------------------------------------

// Проверяет, меньше ли первое число второго.

int Simple\_Is\_Less(s21\_decimal num1, s21\_decimal num2);

// Проверяет, равны ли два числа.

int Simple\_Is\_Equal(s21\_decimal num1, s21\_decimal num2);

// Проверяет, меньше ли или равно первое число второму.

int Simple\_Is\_Less\_Or\_Equal(s21\_decimal num1, s21\_decimal num2);

// Проверяет, больше ли первое число второго.

int Simple\_Is\_Greater(s21\_decimal num1, s21\_decimal num2);

// Проверяет, больше ли или равно первое число второму.

int Simple\_Is\_Greater\_Or\_Equal(s21\_decimal num1, s21\_decimal num2);

// Проверяет, не равны ли два числа.

int Simple\_Is\_Not\_Equal(s21\_decimal num1, s21\_decimal num2);

// ------------------------------------

// Вспомогательные функции

// ------------------------------------

//Округляет число типа s21\_decimal до указанной точности.

void Bank\_Round(s21\_decimal \*value, int count);

// Приводит два числа типа s21\_decimal к одинаковому масштабу.

void Normalize(s21\_decimal \*num1, s21\_decimal \*num2);

**Реализация Функций**

В первую очередь надо реализовать арифметические операции.

*Функция Normalize*

Эта функция выравнивает масштабы двух чисел перед выполнением арифметических операций. Она сравнивает шкалы двух чисел (scale1 и scale2) и приводит их к одному значению. Если одна из шкал меньше другой, то меньшее число домножается на 10 до тех пор, пока обе шкалы не станут равными. В случае ошибки округления, операция прерывается и выполняется банковское округление для большего числа.

void Normalize(s21\_decimal\* num1, s21\_decimal\* num2) {

int scale1 = Get\_Scale(\*num1);

int scale2 = Get\_Scale(\*num2);

int temp1 = num1->bits[SCALE];

int temp2 = num2->bits[SCALE];

int min\_scale = (scale1 < scale2) ? scale1 : scale2;

int max\_scale = (scale1 > scale2) ? scale1 : scale2;

s21\_decimal\* min\_value = (scale1 < scale2) ? num1 : num2;

s21\_decimal\* max\_value = (scale1 > scale2) ? num1 : num2;

s21\_decimal ten;

s21\_from\_int\_to\_decimal(10, &ten);

while (min\_scale != max\_scale) {

if (min\_scale < MAX\_POWER && !Simple\_Mul(\*min\_value, ten, min\_value)) {

min\_scale++;

Set\_Scale(min\_value, min\_scale);

} else {

Bank\_Round(max\_value, max\_scale - min\_scale);

break;

}

}

num1->bits[SCALE] = temp1;

num2->bits[SCALE] = temp2;

Set\_Scale(min\_value, min\_scale);

Set\_Scale(max\_value, min\_scale);

}

*Функция s21\_add*

Функция выполняет сложение двух чисел типа s21\_decimal. Сначала вызывается функция нормализации Normalize, чтобы привести оба числа к одинаковому масштабу. Затем проверяется знак чисел: если знаки одинаковые, происходит простое сложение через вызов функции Simple\_Add, а результат получает тот же знак, что и исходные числа. Если знаки разные, то выполняется вычитание одного числа из другого, и результату присваивается соответствующий знак. После этого производится проверка результата на переполнение/недостаток и при необходимости корректируется масштаб.

int s21\_add(s21\_decimal value1, s21\_decimal value2, s21\_decimal\* result) {

ARITHMETHIC\_RES res = ARITHMETHIC\_OK;

Normalize(&value1, &value2);

int scale1 = Get\_Scale(value1);

int sign1 = Get\_Sign(value1);

int sign2 = Get\_Sign(value2);

if (!(sign1 ^ sign2)) {

res = Simple\_Add(value1, value2, result);

Set\_Sign(result, sign1);

if (res) {

if (sign1)

res = ARITHMETHIC\_SMALL\_INFINITY;

else

res = ARITHMETHIC\_BIG\_INFINITY;

}

} else if (Simple\_Is\_Less\_Or\_Equal(value1, value2)) {

Simple\_Sub(value2, value1, result);

Set\_Sign(result, sign2);

} else {

Simple\_Sub(value1, value2, result);

Set\_Sign(result, sign1);

}

if (res && scale1 > 0) {

Bank\_Round(&value1, 1);

Bank\_Round(&value2, 1);

res = s21\_add(value1, value2, result);

} else

Set\_Scale(result, scale1);

return res;

}

*Функции s21\_sub, s21\_mul, s21\_div*

Функции выполняет операции вычитания, умножения и деления двух чисел типа decimal.

int s21\_sub(s21\_decimal value1, s21\_decimal value2, s21\_decimal\* result) {

ARITHMETHIC\_RES res = ARITHMETHIC\_OK;

Set\_Sign(&value2, !Get\_Sign(value2));

res = s21\_add(value1, value2, result);

return res;

}

int s21\_mul(s21\_decimal value1, s21\_decimal value2, s21\_decimal\* result) {

ARITHMETHIC\_RES res = Simple\_Mul(value1, value2, result);

return res;

}

int s21\_div(s21\_decimal value1, s21\_decimal value2, s21\_decimal\* result) {

s21\_decimal null\_decimal = {0};

if (s21\_is\_equal(value2, null\_decimal)) return ARITHMETHIC\_ZERO\_DIV;

Simple\_Div(value1, value2, result);

return ARITHMETHIC\_OK;

}

Далее стоит реализовать операции для сравнения двух чисел типа decimal.

s21\_is\_less – проверяет, является ли первое число строго меньше второго. Если знаки чисел различаются, результат определяется сразу по знакам. Если знаки совпадают, числа приводятся к общему масштабу функцией Normalize, и тогда используется Simple\_Is\_Less для сравнения.

s21\_is\_equal – проверяет, равны ли два числа. Если знаки различны, числа считаются неравными, кроме случая, когда оба числа равны нулю. Если знаки совпадают, числа нормализуются, и используется Simple\_Is\_Equal для проверки равенства.

s21\_is\_less\_or\_equal – проверяет, является ли первое число меньше или равно второму. Это достигается комбинацией функций s21\_is\_less и s21\_is\_equal.

s21\_is\_greater – проверяет, является ли первое число больше второго. Это реализуется через инверсию результата функции s21\_is\_less\_or\_equal.s

21\_is\_greater\_or\_equal – проверяет, является ли первое число больше или равно второму. Это достигается комбинированием функций s21\_is\_greater и s21\_is\_equal.s21\_is\_not\_equal – проверяет, отличаются ли два числа друг от друга. Это делается через инверсию результата функции s21\_is\_equal.

int s21\_is\_less(s21\_decimal num1, s21\_decimal num2) {

int res;

int sign\_1 = Get\_Sign(num1);

int sign\_2 = Get\_Sign(num2);

if (!sign\_1 && !sign\_2) {

Normalize(&num1, &num2);

res = Simple\_Is\_Less(num1, num2);

} else if (sign\_1 ^ sign\_2) {

res = (sign\_1) ? TRUE : FALSE;

} else {

Normalize(&num1, &num2);

res = !Simple\_Is\_Less(num1, num2);

}

return res;

}

int s21\_is\_equal(s21\_decimal num1, s21\_decimal num2) {

int res;

int sign\_1 = Get\_Sign(num1);

int sign\_2 = Get\_Sign(num2);

s21\_decimal null\_decimal;

s21\_from\_int\_to\_decimal(0, &null\_decimal);

if (sign\_1 ^ sign\_2) {

res = FALSE;

if (Simple\_Is\_Equal(num1, null\_decimal) &&

Simple\_Is\_Equal(num2, null\_decimal)) {

res = TRUE;

}

} else {

Normalize(&num1, &num2);

res = Simple\_Is\_Equal(num1, num2);

}

return res;

}

int s21\_is\_less\_or\_equal(s21\_decimal num1, s21\_decimal num2) {

return s21\_is\_less(num1, num2) || s21\_is\_equal(num1, num2);

}

int s21\_is\_greater(s21\_decimal num1, s21\_decimal num2) {

return !s21\_is\_less\_or\_equal(num1, num2);

}

int s21\_is\_greater\_or\_equal(s21\_decimal num1, s21\_decimal num2) {

return s21\_is\_greater(num1, num2) || s21\_is\_equal(num1, num2);

}

int s21\_is\_not\_equal(s21\_decimal num1, s21\_decimal num2) {

return !s21\_is\_equal(num1, num2);

}

Некоторые функции реализованы через другие, потому что это позволяет избежать дублирования кода и логики. Например, вместо того чтобы заново писать сложные алгоритмы для определения, является ли одно число больше другого, достаточно использовать уже существующие функции для проверки на "меньше или равно" и "равно".

Функции конвертеры.

s21\_from\_int\_to\_decimal

Преобразует целое число (int) в число типа s21\_decimal. Процесс включает следующие шаги:Инициализация структуры s21\_decimal с помощью Init\_Decimal\_Num.Запись абсолютного значения целого числа в младшие биты структуры s21\_decimal.Установка знака числа в зависимости от исходного целого числа.

int s21\_from\_int\_to\_decimal(int src, s21\_decimal \*dst) {

Init\_Decimal\_Num(dst);

dst->bits[LOW\_BIT] = (src >= 0) ? src : -src;

Set\_Sign(dst, src < 0);

return OK;

}

count\_str\_float

Вспомогательная функция для подсчета количества значимых цифр в строке, представляющей вещественное число с плавающей точкой. Эта функция обрезает лишние нули в конце строки и находит позицию десятичной точки.

static int count\_str\_float(float src, char \*str\_src) {

int count\_str = 0, flag = 1;

char str[BUFF\_SIZE];

snprintf(str, sizeof(str), "%f", src);

for (int i = (int)strlen(str) - 1; i >= 0; i--) {

if (str[i] == '0' && flag == 1) {

str[i] = '\0';

continue;

} else

flag = -1;

if (str[i] == '.') break;

count\_str++;

}

strncpy(str\_src, str, strlen(str) + 1);

return count\_str;

}

s21\_from\_float\_to\_decimal

Преобразует вещественное число с плавающей точкой (float) в число типа s21\_decimal. Алгоритм работает следующим образом:Инициализирует структуру s21\_decimal с помощью Init\_Decimal\_Num.Преобразует значение float в строку и удаляет незначащие нули.Проходит по каждому символу строки и добавляет соответствующие цифры к результату.Устанавливает правильный знак и масштаб для полученного числа.

int s21\_from\_float\_to\_decimal(float src, s21\_decimal \*dst) {

Init\_Decimal\_Num(dst);

if (fabsf(src) < 1e-28 && fabsf(src) > 0) return ERROR;

char str\_src[BUFF\_SIZE];

int count\_str = count\_str\_float(src, str\_src), is\_overfull = 0;

s21\_decimal ten;

s21\_from\_int\_to\_decimal(10, &ten);

for (size\_t i = 0; i < strlen(str\_src); i++) {

if (str\_src[i] != '.' && str\_src[i] != '-') {

s21\_decimal add;

s21\_from\_int\_to\_decimal(str\_src[i] - '0', &add);

Simple\_Add(\*dst, add, dst);

is\_overfull = s21\_mul(\*dst, ten, dst);

}

}

if (!is\_overfull) Simple\_Div(\*dst, ten, dst);

Set\_Sign(dst, src < 0);

Set\_Scale(dst, count\_str);

return OK;

}

s21\_from\_decimal\_to\_int

Преобразует число типа s21\_decimal в целое число (int). Этот процесс включает:Деление числа на 10 до тех пор, пока его масштаб не станет нулевым.Сбрасывание старшего бита числа, чтобы получить 32-битное представление.Запись результата в переменную типа int с учетом знака.

int s21\_from\_decimal\_to\_int(s21\_decimal src, int \*dst) {

if (!dst) return ERROR;

int exp = Get\_Scale(src);

s21\_decimal ten;

s21\_from\_int\_to\_decimal(10, &ten);

while (exp > 0) {

Simple\_Div(src, ten, &src);

exp--;

}

Set\_Bit(&src, HIGHEST\_BIT\_IN\_32\_BIT\_PART, 0);

\*dst = src.bits[LOW\_BIT];

int sign = (Get\_Sign(src)) ? -1 : 1;

\*dst \*= sign;

return OK;

}

s21\_from\_decimal\_to\_float

Преобразует число типа s21\_decimal в вещественное число с плавающей точкой (float). Алгоритм состоит из нескольких шагов:Разделение числа на целую часть и дробную часть.Вычисление целой части как суммы степеней двойки.Вычисление дробной части аналогично целой, но с последующим делением на степень 10, соответствующую масштабу числа.Объединение целой и дробной частей и установка правильного знака.

int s21\_from\_decimal\_to\_float(s21\_decimal src, float \*dst) {

int sign\_flag = Get\_Sign(src);

long double temp = 0;

s21\_decimal res = {0}, ten = {0};

s21\_from\_int\_to\_decimal(10, &ten);

s21\_truncate(src, &res);

int sign;

for (int i = 0; i < MAX\_HIGHT\_BIT + 1; i++) {

sign = Get\_Bit(res, i);

temp += sign \* pow(2, i);

}

s21\_decimal float\_part = {0};

s21\_sub(src, res, &float\_part);

int scale = Get\_Scale(src);

long double lf\_part = 0;

int count = 0;

for (int i = 0; i < MAX\_HIGHT\_BIT + 1; i++) {

sign = Get\_Bit(float\_part, i);

count++;

lf\_part += sign \* pow(2, i);

}

\*dst = lf\_part / pow(10, scale) + temp;

if (sign\_flag) \*dst = -(\*dst);

return OK;

}

Другие функции

s21\_floor

int s21\_floor(s21\_decimal value, s21\_decimal \*result) {

s21\_decimal base = {0}, zero = {0}, one = {0};

s21\_from\_int\_to\_decimal(10, &base);

s21\_from\_int\_to\_decimal(0, &zero);

s21\_from\_int\_to\_decimal(1, &one);

int sign = Get\_Sign(value);

s21\_truncate(value, result);

if (sign == 1 && !s21\_is\_equal(value, \*result))

Simple\_Add(\*result, one, result);

Set\_Sign(result, sign);

return OK;

}

Функция s21\_floor округляет число вниз до ближайшего целого. Она использует вспомогательные функции для инициализации чисел 10, 0 и 1 в формате s21\_decimal. Затем она применяет функцию s21\_truncate для удаления дробной части числа. Если число было отрицательным и результат отличается от исходного числа, добавляется единица, чтобы правильно округлить вниз.

s21\_round

int s21\_round(s21\_decimal value, s21\_decimal \*result) {

s21\_decimal base = {0}, mul = {0}, one = {0};

s21\_decimal value\_copy = value;

s21\_truncate(value, result);

s21\_from\_int\_to\_decimal(10, &base);

s21\_from\_int\_to\_decimal(1, &one);

int sign = Get\_Sign(value);

int scale = Get\_Scale(value);

if (scale != 0) scale--;

while (scale != 0) {

s21\_div(value, base, &value);

scale--;

}

mul = Simple\_Div(value, base, &value);

if (mul.bits[LOW\_BIT] >= 5 && !s21\_is\_equal(value\_copy, \*result))

Simple\_Add(\*result, one, result);

Set\_Sign(result, sign);

return OK;

}

Функция s21\_round округляет число до ближайшего целого. Она сначала копирует исходное число и применяет функцию s21\_truncate для получения целой части. Затем она делит число на 10 до тех пор, пока не останется только первая цифра после запятой. Если эта цифра больше или равна 5, то к результату добавляется единица для округления вверх.

s21\_truncate

int s21\_truncate(s21\_decimal value, s21\_decimal \*result) {

s21\_decimal base = {0};

\*result = value;

int sign = Get\_Sign(value);

int scale = Get\_Scale(value);

s21\_from\_int\_to\_decimal(10, &base);

while (scale != 0) {

Simple\_Div(\*result, base, result);

scale--;

}

Set\_Sign(result, sign);

return OK;

}

Функция s21\_truncate удаляет дробную часть числа. Она делит число на 10 столько раз, сколько требуется для уменьшения масштаба числа до нуля, тем самым оставляя только целую часть.

s21\_negate

int s21\_negate(s21\_decimal value, s21\_decimal \*result) {

\*result = value;

Set\_Sign(result, !Get\_Sign(value));

return OK;

}

Функция s21\_negate изменяет знак числа на противоположный. Она просто инвертирует бит знака в структуре s21\_decimal.

Манипуляция битами

Get\_Sign

int Get\_Sign(s21\_decimal number) {

return ((number.bits[SCALE] & (1 << HIGHEST\_BIT\_IN\_32\_BIT\_PART)) == 0) ? 0

: 1;

}

Возвращает знак числа. Если старший бит установлен, то число отрицательно, иначе положительно.

Set\_Sign

void Set\_Sign(s21\_decimal \*number, int sign) {

if (sign == 0) {

number->bits[SCALE] =

number->bits[SCALE] & ~(1 << HIGHEST\_BIT\_IN\_32\_BIT\_PART);

} else {

number->bits[SCALE] =

number->bits[SCALE] | (1 << HIGHEST\_BIT\_IN\_32\_BIT\_PART);

}

}

Устанавливает знак числа. Если передается 0, то число становится положительным, если 1 — отрицательным.

Get\_Scale

int Get\_Scale(s21\_decimal number) {

int shift = START\_SCALE\_BIT;

int scale = 0, i = 0;

while (shift <= LAST\_SCALE\_BIT) {

int bit = ((number.bits[SCALE] & (1 << shift)) != 0);

scale += bit \* pow(2, i);

shift++;

i++;

}

return scale;

}

Получает масштаб числа (количество знаков после запятой).

Set\_Scale

void Set\_Scale(s21\_decimal \*number, int scale) {

int shift = START\_SCALE\_BIT;

while (shift <= LAST\_SCALE\_BIT) {

if (scale > 0 && scale % 2 == 1)

number->bits[SCALE] = number->bits[SCALE] | (1 << shift);

else

number->bits[SCALE] = number->bits[SCALE] & ~(1 << shift);

scale /= 2;

shift++;

}

}

Устанавливает новый масштаб числа.

Get\_Bit

int Get\_Bit(s21\_decimal number, int pos) {

return ((number.bits[pos / COUNT\_BITS\_IN\_PART] &

(1 << pos % COUNT\_BITS\_IN\_PART)) == 0)

? 0

: 1;

}

Возвращает значение бита в заданной позиции.

Set\_Bit

void Set\_Bit(s21\_decimal \*number, int bit, int sign) {

int pos = bit / COUNT\_BITS\_IN\_PART;

if (sign)

number->bits[pos] = number->bits[pos] | (1 << bit % COUNT\_BITS\_IN\_PART);

else

number->bits[pos] = number->bits[pos] & ~(1 << bit % COUNT\_BITS\_IN\_PART);

}

Устанавливает бит в заданной позиции в соответствии со значением sign.

shift\_left

int shift\_left(s21\_decimal \*number) {

int is\_overflow = Get\_Bit(\*number, MAX\_HIGHT\_BIT);

for (int i = MAX\_HIGHT\_BIT; i >= 0; i--) {

if (i != 0)

Set\_Bit(number, i, Get\_Bit(\*number, i - 1));

else

Set\_Bit(number, i, 0);

}

return is\_overflow;

}

Сдвигает все биты числа влево на одну позицию. Возвращает 1, если произошел выход за пределы допустимого диапазона.

Shift\_Right

int Shift\_Right(s21\_decimal \*number) {

int is\_overflow = Get\_Bit(\*number, 1);

for (int i = 0; i < MAX\_HIGHT\_BIT + 1; i++) {

if (i != MAX\_HIGHT\_BIT)

Set\_Bit(number, i, Get\_Bit(\*number, i + 1));

else

Set\_Bit(number, i, 0);

}

return is\_overflow;

}

Сдвигает все биты числа вправо на одну позицию. Возвращает 1, если произошло смещение младшего бита.

Инициализация

Init\_Decimal\_Num

void Init\_Decimal\_Num(s21\_decimal \*number) {

memset(number, 0, sizeof(s21\_decimal));

}

Инициализирует структуру s21\_decimal, устанавливая все её поля в 0.

Простые арифметические операции

Simple\_Add

int Simple\_Add(s21\_decimal value1, s21\_decimal value2, s21\_decimal \*result) {

Init\_Decimal\_Num(result);

int rank = 0;

for (int i = 0; i < MAX\_HIGHT\_BIT + 1; i++) {

int bit\_dec1 = Get\_Bit(value1, i);

int bit\_dec2 = Get\_Bit(value2, i);

Set\_Bit(result, i, bit\_dec1 ^ bit\_dec2 ^ rank);

rank = (bit\_dec1 && bit\_dec2) || (bit\_dec1 && rank) || (bit\_dec2 && rank);

}

return rank;

}

Выполняет побитовое сложение двух чисел. Возвращает 1, если произошла потеря точности (переполнение).

Simple\_Sub

void Simple\_Sub(s21\_decimal value1, s21\_decimal value2, s21\_decimal \*result) {

Init\_Decimal\_Num(result);

for (int i = 0; i < MAX\_HIGHT\_BIT + 1; i++) {

int bit\_dec1 = Get\_Bit(value1, i);

int bit\_dec2 = Get\_Bit(value2, i);

Set\_Bit(result, i, bit\_dec1 ^ bit\_dec2);

if (!bit\_dec1 && bit\_dec2) {

int j = i + 1;

while ((bit\_dec1 = Get\_Bit(value1, j)) != 1) {

Set\_Bit(&value1, j, 1);

j++;

}

Set\_Bit(&value1, j, 0);

}

}

}

Выполняет побитовое вычитание первого числа из второго.Simple\_Mul

int Simple\_Mul(s21\_decimal value1, s21\_decimal value2, s21\_decimal \*result) {

s21\_decimal tmp;

Init\_Decimal\_Num(&tmp);

int is\_overflow = 0;

for (int i = 0; i < MAX\_HIGHT\_BIT + 1 && !is\_overflow; i++) {

if (Get\_Bit(value2, i) == 1) {

s21\_decimal temp = value1;

int j = 0;

while (j < i) {

if (shift\_left(&temp)) {

is\_overflow = 1;

break;

}

j++;

}

if (is\_overflow || (is\_overflow = Simple\_Add(temp, tmp, &tmp))) {

break;

}

}

}

if (!is\_overflow) \*result = tmp;

return is\_overflow;

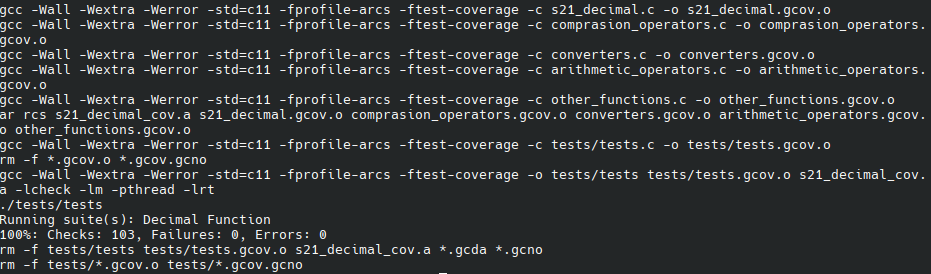
}

Выполняет умножение двух чисел. Возвращает 1, если произошла потеря точности (переполнение).

**Тестирование**

Тестирование представляет собой важный этап процесса разработки программного обеспечения, который направлен на проверку корректности работы программы и её соответствие установленным требованиям. В ходе тестирования выявляются ошибки, баги и уязвимости, способные негативно сказаться на качестве конечного продукта. Это позволяет повысить надёжность системы, улучшить взаимодействие с пользователем и снизить возможные риски при эксплуатации ПО. Быстрое обнаружение и исправление проблем существенно уменьшает затраты на дальнейшую доработку и техническую поддержку.

Помимо этого, тестирование помогает оценить производительность системы, её устойчивость к высокой нагрузке и способность функционировать в разных условиях. Тестирование также играет ключевую роль в защите данных, предотвращая возможные угрозы и утечки информации. Следовательно, тестирование является неотъемлемой частью успешной разработки и внедрения высококачественного программного обеспечения.

Проведя тестирование своего проекта, я убедился в правильности выполненной работы, так как все тесты, написанные мной были выполненны идеально.

**Вывод**

В ходе реализации типа decimal на языке программирования Си были разработаны арифметические и сравнительные функции, а также другие важные аспекты работы с десятичными числами. Этот проект позволил мне углубить знания в области низкоуровневого программирования и понять принципы работы с битовыми операциями и структурными данными.

Во-первых, я научился эффективно работать с памятью, оптимизировать использование ресурсов и улучшать производительность программы. Разработка собственной структуры для хранения десятичных чисел позволила лучше понять внутренние механизмы компилятора и особенности архитектуры процессора.

Во-вторых, благодаря созданию арифметических функций я освоил методы точного выполнения операций сложения, вычитания, умножения и деления, избегая ошибок округления и переполнения. Это дало мне возможность глубже изучить алгоритмы обработки больших чисел и способы их оптимизации.

Кроме того, разработка сравнительных функций помогла мне освоить принципы сортировки и поиска, а также научиться правильно интерпретировать результаты сравнения чисел различной длины и точности.

Наконец, этот проект дал мне опыт работы с документированием кода и созданием тестов для проверки правильности функционирования программных модулей. Я понял важность тестирования и отладки, особенно когда речь идет о работе с числовыми данными, требующими высокой точности.

Таким образом, реализация типа decimal стала ценным опытом, позволившим мне улучшить навыки программирования на языке Си и приобрести новые знания в области работы с числовыми данными.