|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** ***ИУК «Информатика и управление»***

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ДОМАШНЯЯ РАБОТА №1**

**«Моделирование операций над длинными числами»**

**ДИСЦИПЛИНА: «Типы и структуры данных»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-42Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Пчелинцева Н.И. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2022

**Цель:** формирование практических навыков моделирования операций над длинными числами.

**Задание:**

1. Познакомиться с представлением чисел в памяти компьютера.
2. Создать собственную модель для представления длинного числа в памяти компьютера.
3. Научиться составлять и реализовывать алгоритмы для арифметических операций над длинными числами.
4. Смоделировать математическую операцию с длинными числами согласно варианту.

**Вариант 7**

Смоделировать операцию деления двух действительных чисел в форме (zn)m.n Е N, где суммарная длина мантиссы (m+n) – до 30 значащих цифр, а величина порядка N – до 5 цифр. Результат выдать в форме (zn)0.m1 Е N1.

**Листинг:**

***Real.h***

#pragma once

#include <string>

namespace Real

{

class Real

{

public:

Real();

Real(std::string number);

Real(Real& real);

~Real();

std::string ToString();;

Real\* Divide(Real number);

private:

void IncDegree();

void DecDegree();

void DecreaseDegree(int\* value, unsigned short size);

void IncreaseDegree(int\* value, unsigned short size);

short CompareDegree(int\* value, unsigned short size);

int\* CopyDegree();

static const struct Constants

{

Constants()

{

for (unsigned short i{}; i < MaxDegreeLength - 1; ++i)

DegreeShift[i] = Base - 1;

DegreeShift[MaxDegreeLength - 1] = (Base - 1) / 2;

}

static const unsigned short MaxMantissaLength = 30;

static const unsigned short MaxDegreeLength = 5;

static const unsigned short Base = 10;

int DegreeShift[MaxDegreeLength];

} \_Constants;

bool \_sign{};

int\* \_mantissa{};

int\* \_degree{};

};

class DivisionByZeroException

{

public:

DivisionByZeroException(std::string error);

std::string GetError();

private:

std::string \_error{};

};

}

***Real.cpp***

#include "Real.h"

namespace Real

{

Real::Constants const Real::\_Constants;

Real::Real() : \_sign(false), \_mantissa(new int[\_Constants.MaxMantissaLength]{}),

\_degree(new int[\_Constants.MaxDegreeLength]{}) {}

Real::Real(std::string number)

{

\_mantissa = new int[\_Constants.MaxMantissaLength] {};

\_degree = new int[\_Constants.MaxDegreeLength] {};

for (unsigned short i{}; i < \_Constants.MaxDegreeLength; ++i)

\_degree[i] = \_Constants.DegreeShift[i];

unsigned short\* temp = new unsigned short[\_Constants.MaxMantissaLength] {};

int t = 0;

bool isPointReached = false;

bool isNotZeroReached = false;

for (unsigned short i{}; i < number.length(); ++i)

{

if ('1' <= number[i] && number[i] <= '9')

{

temp[t] = number[i] - '0';

isNotZeroReached = true;

if (!isPointReached)

IncDegree();

++t;

}

else if (number[i] == '0')

{

temp[t] = 0;

if (isPointReached && !isNotZeroReached)

DecDegree();

else if (!isPointReached && isNotZeroReached)

IncDegree();

++t;

}

else if (number[i] == '-')

\_sign = true;

else if (number[i] == '.')

isPointReached = true;

}

if (isNotZeroReached)

{

unsigned short i = 0;

--t;

while (temp[t] == 0)

--t;

while (t >= 0)

{

\_mantissa[i] = temp[t];

--t;

++i;

}

DecDegree();

}

else

\_sign = false;

delete[] temp;

}

Real::Real(Real& real) : \_sign(real.\_sign)

{

\_degree = new int[\_Constants.MaxDegreeLength] {};

\_mantissa = new int[\_Constants.MaxMantissaLength] {};

for (unsigned short i{}; i < \_Constants.MaxDegreeLength; ++i)

\_degree[i] = real.\_degree[i];

for (unsigned short i{}; i < \_Constants.MaxMantissaLength; ++i)

\_mantissa[i] = real.\_mantissa[i];

}

Real::~Real()

{

delete[] \_mantissa;

delete[] \_degree;

}

std::string Real::ToString()

{

std::string number = "";

if (\_sign)

number += '-';

int pos = -1;

for (int i = \_Constants.MaxMantissaLength - 1; i >= 0; --i)

if (\_mantissa[i] != 0)

{

pos = i;

break;

}

if (pos == -1)

{

number += "0.0 E 0";

return number;

}

else

{

number += "0.";

for (; pos >= 0; --pos)

{

number += (\_mantissa[pos] + '0');

}

}

number += " E ";

int\* temp = CopyDegree();

IncDegree();

if (CompareDegree((int\*)\_Constants.DegreeShift, \_Constants.MaxDegreeLength) == 1)

{

number += "-";

for (unsigned short i{}; i < \_Constants.MaxDegreeLength; ++i)

\_degree[i] = \_Constants.DegreeShift[i];

DecreaseDegree(temp, \_Constants.MaxDegreeLength);

DecDegree();

int i = \_Constants.MaxDegreeLength - 1;

while (\_degree[i] == 0)

--i;

for (; i >= 0; --i)

number += (\_degree[i] + '0');

}

else if (CompareDegree((int\*)\_Constants.DegreeShift, \_Constants.MaxDegreeLength) == -1)

{

DecreaseDegree((int\*)\_Constants.DegreeShift, \_Constants.MaxDegreeLength);

int i = \_Constants.MaxDegreeLength - 1;

while (\_degree[i] == 0)

--i;

for (; i >= 0; --i)

number += (\_degree[i] + '0');

}

else

number += '0';

delete[] \_degree;

\_degree = temp;

return number;

}

Real\* Real::Divide(Real number)

{

bool dividedByZero = true;

for (unsigned short i{}; i < \_Constants.MaxMantissaLength; ++i)

if (number.\_mantissa[i] != 0)

{

dividedByZero = false;

break;

}

if (dividedByZero)

throw DivisionByZeroException("Division by zero");

bool divisibleIsZero = true;

for (unsigned short i{}; i < \_Constants.MaxMantissaLength; ++i)

if (\_mantissa[i] != 0)

{

divisibleIsZero = false;

break;

}

if (divisibleIsZero)

return new Real();

int\* degree{};

if (CompareDegree(number.\_degree, \_Constants.MaxDegreeLength) == -1)

{

int\* temp = CopyDegree();

DecreaseDegree(number.\_degree, \_Constants.MaxDegreeLength);

IncreaseDegree((int\*)\_Constants.DegreeShift, \_Constants.MaxDegreeLength);

degree = CopyDegree();

\_degree = temp;

}

else if (CompareDegree(number.\_degree, \_Constants.MaxDegreeLength) == 1)

{

number.DecreaseDegree(\_degree, \_Constants.MaxDegreeLength);

number.IncreaseDegree((int\*)\_Constants.DegreeShift, \_Constants.MaxDegreeLength);

degree = number.CopyDegree();

}

else

{

degree = new int[\_Constants.MaxDegreeLength] {};

for (unsigned short i{}; i < \_Constants.MaxDegreeLength; ++i)

degree[i] = \_Constants.DegreeShift[i];

}

unsigned short divisibleMantissaSize = \_Constants.MaxMantissaLength;

for (int i = \_Constants.MaxMantissaLength - 1; i >= 0; --i)

if (\_mantissa[i] != 0)

break;

else

--divisibleMantissaSize;

unsigned short divisorMantissaSize = \_Constants.MaxMantissaLength;

for (int i = \_Constants.MaxMantissaLength - 1; i >= 0; --i)

if (number.\_mantissa[i] != 0)

break;

else

--divisorMantissaSize;

int\* divisibleMantissa{};

if (divisibleMantissaSize < divisorMantissaSize)

{

divisibleMantissa = new int[divisorMantissaSize] {};

for (unsigned short i{}; i < divisibleMantissaSize; ++i)

divisibleMantissa[i + divisorMantissaSize - divisibleMantissaSize] = \_mantissa[i];

divisibleMantissaSize = divisorMantissaSize;

}

else

{

divisibleMantissa = new int[divisibleMantissaSize] {};

for (unsigned short i{}; i < divisibleMantissaSize; ++i)

divisibleMantissa[i] = \_mantissa[i];

}

int\* num = new int[divisorMantissaSize] {};

for (unsigned short i{}; i < divisorMantissaSize; ++i)

num[divisorMantissaSize - i - 1] = divisibleMantissa[divisibleMantissaSize - i - 1];

unsigned short numSize = divisorMantissaSize;

unsigned short digitPosition = divisorMantissaSize;

bool needExpansion = false;

for (int i = divisorMantissaSize - 1; i >= 0; --i)

if (num[i] < number.\_mantissa[i])

{

needExpansion = true;

break;

}

else if (num[i] > number.\_mantissa[i])

break;

if (needExpansion)

{

int\* temp = new int[numSize + 1]{};

for (unsigned short i{}; i < numSize; ++i)

temp[i + 1] = num[i];

delete[] num;

num = temp;

++numSize;

if (digitPosition != divisibleMantissaSize)

num[0] = divisibleMantissa[divisibleMantissaSize - digitPosition - 1];

unsigned short i = 0;

--degree[i];

while (i < \_Constants.MaxDegreeLength && degree[i] < 0)

{

degree[i] = \_Constants.Base - 1;

++i;

if (i < \_Constants.MaxDegreeLength)

--degree[i];

}

++digitPosition;

}

unsigned short counter = \_Constants.MaxMantissaLength;

int\* result{};

unsigned short resultSize = 0;

bool numIsZero = false;

while (!numIsZero && counter > 0)

{

bool isNumLess = false;

if (numSize < divisorMantissaSize)

isNumLess = true;

else if (numSize == divisorMantissaSize)

for (int i = divisorMantissaSize - 1; i >= 0; --i)

if (num[i] < number.\_mantissa[i])

{

isNumLess = true;

break;

}

else if (num[i] > number.\_mantissa[i])

break;

int\* temp = new int[resultSize + 1]{};

for (unsigned short i{}; i < resultSize; ++i)

temp[i + 1] = result[i];

delete[] result;

result = temp;

++resultSize;

if (!isNumLess)

{

unsigned short k = 1;

bool kIsFound = false;

while (!kIsFound)

{

int\* subtracted = new int[numSize] {};

unsigned short r = 0;

for (unsigned short i{}; i < divisorMantissaSize; ++i)

{

subtracted[i] = (number.\_mantissa[i] \* k + r) % \_Constants.Base;

r = (number.\_mantissa[i] \* k + r) / \_Constants.Base;

}

if (numSize > divisorMantissaSize)

subtracted[numSize - 1] = r;

int\* difference = new int[numSize] {};

r = 0;

for (unsigned short i{}; i < numSize; ++i)

{

difference[i] = num[i] - subtracted[i] - r;

if (difference[i] < 0)

{

r = 1;

difference[i] += \_Constants.Base;

}

else

r = 0;

}

delete[] subtracted;

unsigned short differenceSize = numSize;

for (int i = numSize - 1; i >= 0; --i)

if (difference[i] != 0)

break;

else

--differenceSize;

if (differenceSize > divisorMantissaSize)

++k;

else if (differenceSize < divisorMantissaSize)

kIsFound = true;

else

{

bool isEqual = true;

for (int i = divisorMantissaSize - 1; i >= 0; --i)

if (difference[i] > number.\_mantissa[i])

{

++k;

isEqual = false;

break;

}

else if (difference[i] < number.\_mantissa[i])

{

kIsFound = true;

isEqual = false;

break;

}

if (isEqual)

++k;

}

if (kIsFound)

{

delete[] num;

num = new int[differenceSize] {};

for (unsigned short i{}; i < differenceSize; ++i)

num[i] = difference[i];

numSize = differenceSize;

}

delete[] difference;

}

result[0] = k;

}

temp = new int[numSize + 1]{};

for (unsigned short i{}; i < numSize; ++i)

temp[i + 1] = num[i];

delete[] num;

num = temp;

++numSize;

if (digitPosition < divisibleMantissaSize)

{

num[0] = divisibleMantissa[divisibleMantissaSize - digitPosition - 1];

++digitPosition;

}

--counter;

numIsZero = true;

for (unsigned short i{}; i < numSize; ++i)

if (num[i] != 0)

{

numIsZero = false;

break;

}

}

Real\* resultReal = new Real();

resultReal->\_sign = \_sign && !number.\_sign || !\_sign && number.\_sign;

for (unsigned short i{}; i < resultSize; ++i)

resultReal->\_mantissa[i] = result[i];

delete[] resultReal->\_degree;

resultReal->\_degree = degree;

delete[] result;

delete[] num;

delete[] divisibleMantissa;

return resultReal;

}

void Real::IncDegree()

{

unsigned short i = 0;

++\_degree[i];

while (i < \_Constants.MaxDegreeLength && \_degree[i] == \_Constants.Base)

{

\_degree[i] = 0;

++i;

if (i < \_Constants.MaxDegreeLength)

++\_degree[i];

}

}

void Real::DecDegree()

{

unsigned short i = 0;

--\_degree[i];

while (i < \_Constants.MaxDegreeLength && \_degree[i] < 0)

{

\_degree[i] = \_Constants.Base - 1;

++i;

if (i < \_Constants.MaxDegreeLength)

--\_degree[i];

}

}

void Real::DecreaseDegree(int\* value, unsigned short size)

{

unsigned short r = 0;

for (unsigned short i{}; i < size; ++i)

{

\_degree[i] -= r;

\_degree[i] -= value[i];

if (\_degree[i] < 0)

{

\_degree[i] += \_Constants.Base;

r = 1;

}

else

r = 0;

}

if (size < \_Constants.MaxDegreeLength)

\_degree[size] -= r;

}

void Real::IncreaseDegree(int\* value, unsigned short size)

{

unsigned short r = 0;

for (unsigned short i{}; i < size; ++i)

{

\_degree[i] += r;

\_degree[i] += value[i];

if (\_degree[i] >= \_Constants.Base)

{

\_degree[i] -= \_Constants.Base;

r = 1;

}

else

r = 0;

}

if (size < \_Constants.MaxDegreeLength)

\_degree[size] += r;

}

short Real::CompareDegree(int\* value, unsigned short size)

{

if (size > \_Constants.MaxDegreeLength)

return 1;

else if (size < \_Constants.MaxDegreeLength)

return -1;

else

{

for (int i = \_Constants.MaxDegreeLength - 1; i >= 0; --i)

{

if (value[i] > \_degree[i])

return 1;

else if (value[i] < \_degree[i])

return -1;

}

}

return 0;

}

int\* Real::CopyDegree()

{

int\* temp = new int[\_Constants.MaxDegreeLength] {};

for (unsigned short i{}; i < \_Constants.MaxDegreeLength; ++i)

temp[i] = \_degree[i];

return temp;

}

DivisionByZeroException::DivisionByZeroException(std::string error) : \_error(error) {}

std::string DivisionByZeroException::GetError() { return \_error; }

}

***Main.cpp***

#include "Real.h"

#include <iostream>

int main()

{

std::cout << "Divisible = ";

std::string divisibleInput{};

std::getline(std::cin, divisibleInput);

std::cout << "Divisor = ";

std::string divisorInput{};

std::getline(std::cin, divisorInput);

Real::Real divisible{ divisibleInput };

Real::Real divisor{ divisorInput };

try

{

std::cout << divisible.ToString() << " : " << divisor.ToString()

<< " = " << divisible.Divide(divisor)->ToString() << "\n";

}

catch (Real::DivisionByZeroException& error)

{

std::cout << error.GetError() << "\n";

}

return 0;

}

**Результат:**

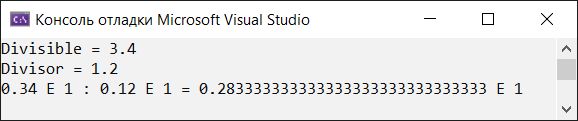


Рис. 1. Пример деления №1

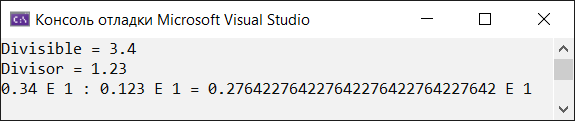


Рис. 2. Пример деления №2

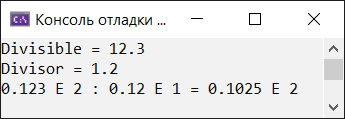


Рис. 3. Пример деления №3

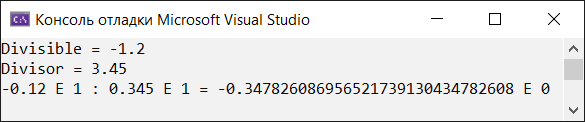


Рис. 4. Пример деления №4

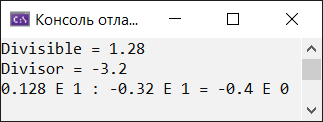


Рис. 5. Пример деления №5

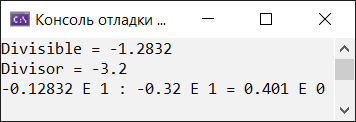


Рис. 6. Пример деления №6

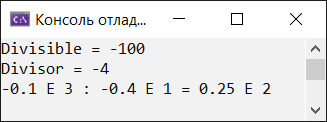


Рис. 7. Пример деления №7

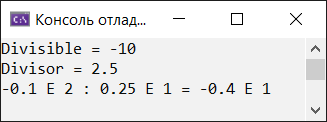


Рис. 8. Пример деления №8

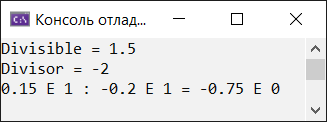


Рис. 9. Пример деления №9

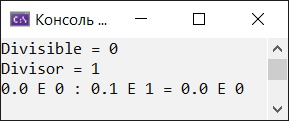


Рис. 10. Пример деления №10

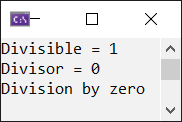


Рис. 11. Пример деления №11

**Вывод:** в ходе выполнения лабораторной работы были получены навыки создания собственного класса вещественных чисел, прототипа кода, сравнения, сложения, вычитания, умножения на цифру или порядок целых чисел, деления вещественных чисел, работы со статическими полями класса.

**ОСНОВНАЯ ЛИТЕРАТУРА**

1. Алексеев В.Е. Графы и алгоритмы. Структуры данных. Модели вычислений [Электронный ресурс]/ В.Е. Алексеев, В.А. Таланов. —Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 153 c. — Режим доступа: <http://www.iprbookshop.ru/52186.html>
2. Вирт Никлаус. Алгоритмы и структуры данных [Электронный ресурс]/ Никлаус Вирт — Электрон. текстовые данные. — Саратов: Профобразование, 2017. — 272 c. — Режим доступа: <http://www.iprbookshop.ru/63821.html>
3. Самуйлов С.В. Алгоритмы и структуры обработки данных [Электронный ресурс]: учебное пособие/ С.В. Самуйлов. — Электрон. текстовые данные. — Саратов: Вузовское образование, 2016. — 132 c.— Режим доступа: http://www.iprbookshop.ru/47275.html

**ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА**

1. Костюкова Н.И. Графы и их применение [Электронный ресурс]/ Н.И. Костюкова. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 147 c. — Режим доступа: <http://www.iprbookshop.ru/52185.html>
2. Сундукова Т.О. Структуры и алгоритмы компьютерной обработки данных [Электронный ресурс]/ Т.О. Сундукова, Г.В. Ваныкина. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 749 c.— Режим доступа: http://www.iprbookshop.ru/57384.html