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. ИУК4-32Б _____ (_____Зудин Д.В._____)
(Подпись) (Ф.И.О.)

Проверил: _____ (___Пчелинцева Н.И.___)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты): 14.12.22

Результаты сдачи (защиты):

- Балльная оценка: 95
- Оценка: хорошо

Калуга, 2022 г.

Цель: формирование практических навыков моделирования операций над длинными числами.

Задачи:

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 №20

Формулировка задания

Создать программу согласно полученному варианту. При выполнении домашней работы запрещается использовать сторонние классы и компоненты, реализующие заявленную функциональность.

Индивидуальное задание

Условные обозначения:

(zn) – знак числа

N, N1 – величина порядка

E – основание числа

Запись числа (zn)0.m E N соответствует следующей записи числа

(zn)0.m*10 .

Смоделировать операцию вычисления среднего арифметического целого числа длиной до 30 десятичных цифр и действительного числа в форме (zn)m.n E 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 E N1.

Листинг файла HugeNumber.h

```
#include <string>
#include <iostream>
#include <regex>
#include <algorithm>

class HugeNumber
{
public:
```

```
	HugeNumber(bool sign = false, std::string mantissa = "0.0", int order =
0);
	HugeNumber(const HugeNumber& hn);

	bool getSign();
	std::string getMantissa();
	int getOrder();
	void ReadHugeNumber();
	void NormalizeNumber();
	void ChangeOrder(int newOrder = 0);

	friend std::ostream& operator<<(std::ostream& out, const HugeNumber&
hn);
	friend bool operator>(const HugeNumber& hn1, const HugeNumber& hn2);
	friend bool operator<(const HugeNumber& hn1, const HugeNumber& hn2);
	friend bool operator==(const HugeNumber& hn1, const HugeNumber& hn2);
	friend bool operator!=(const HugeNumber& hn1, const HugeNumber& hn2);
	friend bool operator>=(const HugeNumber& hn1, const HugeNumber& hn2);
	friend bool operator<=(const HugeNumber& hn1, const HugeNumber& hn2);
	friend HugeNumber operator-(const HugeNumber& hn);
	friend HugeNumber operator/(const HugeNumber& hn, const int n);
	friend HugeNumber operator-(const HugeNumber& hn1, const HugeNumber&
hn2);
	friend HugeNumber operator+(const HugeNumber& hn1, const HugeNumber&
hn2);

private:
	void RemoveExtraZeros();
	int getIntPartLength();
	int getFracPartLength();

	bool sign;                    // Знак числа (false +, true -)
	std::string mantissa;         // Мантисса числа
	int order;                    // Порядок
};
```

Листинг файла HugeNumber.cpp

```
#include "HugeNumber.h"

HugeNumber::HugeNumber(bool sign, std::string mantissa, int order) :
	sign(sign), mantissa(mantissa), order(order)
{
	if (mantissa[0] == '.')
	{
		mantissa = "0" + mantissa;
	}
	if (mantissa.find('.') == std::string::npos)
	{
		mantissa += ".0";
	}
}

HugeNumber::HugeNumber(const HugeNumber& hn) :
	sign(hn.sign), mantissa(hn.mantissa), order(hn.order)
{
}

bool HugeNumber::getSign()
{
	return sign;
}
```

```
std::string HugeNumber::getMantissa()
{
	return mantissa;
}

int HugeNumber::getOrder()
{
	return order;
}

void HugeNumber::ReadHugeNumber()
{
	bool correct = false;
	std::string num{};
	while (!correct)
	{
		std::cout << "Введите число: ";
		correct = true;
		std::getline(std::cin, num);
		static const std::regex r("^[-+]?[0-9]*[.]?[0-9]+(?:[e][-+]?[0-
9]+)?$");
		if (num.length() < 3 || !std::regex_match(num.data(), r))
		{
			std::cout << "Неверная запись числа!" << std::endl;
			correct = false;
		}
	}
	sign = (num[0] == '-');
	size_t eindex = num.find("e");
	mantissa = "";
	for (size_t i = 0; i < eindex; i++)
	{
		if (num[i] != '+' && num[i] != '-')
		{
			mantissa += num[i];
		}
	}
	if (mantissa[0] == '+')
	{
		mantissa.erase(0, 1);
	}
	std::string sorder = "";
	for (size_t i = eindex + 1; i < num.length(); i++)
	{
		sorder += num[i];
	}
	order = std::stoi(sorder);
	if (mantissa.find('.') == std::string::npos)
	{
		mantissa += ".0";
	}
}

std::ostream& operator<<(std::ostream& out, const HugeNumber& hn)
{
	out << (hn.sign ? "-" : "") << hn.mantissa;
	if (hn.order != 0)
	{
		out << "e" << hn.order;
	}
	return out;
}
```

```
void HugeNumber::RemoveExtraZeros()
{
	mantissa = mantissa.substr(mantissa.find_first_not_of('0'));		//
Удаление лишних нулей в начале
	mantissa = mantissa.substr(0, mantissa.find_last_not_of('0') + 1);	//
Удаление лишних нулей в конце
	mantissa = (mantissa[0] == '.' ? "0" : "") + mantissa;
	mantissa += (mantissa[mantissa.length() - 1] == '.' ? "0" : "");
}

void HugeNumber::ChangeOrder(int newOrder)
{
	int delta = order - newOrder;
	for (int i = 0; i <= abs(delta); i++)
	{
		mantissa = "0" + mantissa + "0";
	}
	int indexPoint = mantissa.find('.');
	mantissa.erase(indexPoint, 1);
	mantissa.insert(indexPoint + delta, ".");
	RemoveExtraZeros();
	order = newOrder;
}

bool operator>(const HugeNumber& hn1, const HugeNumber& hn2)
{
	HugeNumber HN1 = hn1;
	HugeNumber HN2 = hn2;
	HN1.ChangeOrder();
	HN2.ChangeOrder();

	if (!HN1.sign && HN2.sign)
	{
		return true;
	}
	if (HN1.sign && !HN2.sign)
	{
		return false;
	}
	bool inverse = false;
	if (HN1.sign && HN2.sign)
	{
		inverse = true;
	}
	if (HN1.mantissa.length() > HN2.mantissa.length())
	{
		return !inverse;
	}
	if (HN1.mantissa.length() < HN2.mantissa.length())
	{
		return inverse;
	}
	for (size_t i = 0; i < HN1.mantissa.length(); i++)
	{
		if (HN1.mantissa[i] > HN2.mantissa[i])
		{
			return !inverse;
		}
		if (HN1.mantissa[i] < HN2.mantissa[i])
		{
			return inverse;
		}
```

```
	}
	return false;
}

bool operator==(const HugeNumber& hn1, const HugeNumber& hn2)
{
	HugeNumber HN1 = hn1;
	HugeNumber HN2 = hn2;
	HN1.ChangeOrder();
	HN2.ChangeOrder();

	if (HN1.sign != HN2.sign)
	{
		return false;
	}
	if (HN1.mantissa.length() != HN2.mantissa.length())
	{
		return false;
	}
	for (size_t i = 0; i < HN1.mantissa.length(); i++)
	{
		if (HN1.mantissa[i] != HN2.mantissa[i])
		{
			return false;
		}
	}
	return true;
}

bool operator!=(const HugeNumber& hn1, const HugeNumber& hn2)
{
	HugeNumber HN1 = hn1;
	HugeNumber HN2 = hn2;
	HN1.ChangeOrder();
	HN2.ChangeOrder();

	return !(HN1 == HN2);
}

bool operator<(const HugeNumber& hn1, const HugeNumber& hn2)
{
	HugeNumber HN1 = hn1;
	HugeNumber HN2 = hn2;
	HN1.ChangeOrder();
	HN2.ChangeOrder();

	return (!(HN1 > HN2)) && (HN1 != HN2);
}

bool operator>=(const HugeNumber& hn1, const HugeNumber& hn2)
{
	HugeNumber HN1 = hn1;
	HugeNumber HN2 = hn2;
	HN1.ChangeOrder();
	HN2.ChangeOrder();

	return (HN1 > HN2) || (HN1 == HN2);
}

bool operator<=(const HugeNumber& hn1, const HugeNumber& hn2)
{
	HugeNumber HN1 = hn1;
	HugeNumber HN2 = hn2;
```

```
	HN1.ChangeOrder();
	HN2.ChangeOrder();

	return (HN1 < HN2) || (HN1 == HN2);
}

HugeNumber operator-(const HugeNumber& hn)
{
	return HugeNumber(!hn.sign, hn.mantissa, hn.order);
}

HugeNumber operator-(const HugeNumber& hn1, const HugeNumber& hn2)
{
	return hn1 + (-hn2);
}

HugeNumber operator+(const HugeNumber& hn1, const HugeNumber& hn2)
{
	HugeNumber HN1 = hn1;
	HugeNumber HN2 = hn2;
	HN1.ChangeOrder();
	HN2.ChangeOrder();

	if (HN1 == (-HN2))
	{
		return HugeNumber();
	}
	if (HN1.sign && HN2.sign)
	{
		return -((-HN1) + (-HN2));
	}
	if (!HN1.sign && HN2.sign && (HN1 < (-HN2)))
	{
		return -((-HN2) - HN1);
	}
	if (HN1.sign && !HN2.sign)
	{
		if (HN2 < (-HN1))
		{
			return -((-HN1) - HN2);
		}
		return HN2 + HN1;
	}

	const int IntPartLength1 = HN1.getIntPartLength();
	const int IntPartLength2 = HN2.getIntPartLength();
	const int FracPartLength1 = HN1.getFracPartLength();
	const int FracPartLength2 = HN2.getFracPartLength();
	const int extraIntZerosLength = abs(IntPartLength1 - IntPartLength2);
	const int extraFracZerosLength = abs(FracPartLength1 - FracPartLength2);

	for (int i = 0; i < extraIntZerosLength; i++)
	{
		if (IntPartLength1 < IntPartLength2)
		{
			HN1.mantissa = "0" + HN1.mantissa;
		}
		else
		{
			HN2.mantissa = "0" + HN2.mantissa;
		}
	}
	for (int i = 0; i < extraFracZerosLength; i++)
```

```
	{
		if (FracPartLength1 < FracPartLength2)
		{
			HN1.mantissa += "0";
		}
		else
		{
			HN2.mantissa += "0";
		}
	}
	HN1.mantissa = "0" + HN1.mantissa;
	HN2.mantissa = "0" + HN2.mantissa;
	std::reverse(HN1.mantissa.begin(), HN1.mantissa.end());
	std::reverse(HN2.mantissa.begin(), HN2.mantissa.end());
	int indexPoint = HN1.mantissa.find('.');
	HN1.mantissa.erase(indexPoint, 1);
	HN2.mantissa.erase(indexPoint, 1);

	std::string result = "";
	int mind = 0;
	if (!HN2.sign)
	{
		for (int i = 0; i < HN1.mantissa.length(); i++)
		{
			int digit1 = HN1.mantissa[i] - '0';
			int digit2 = HN2.mantissa[i] - '0';
			int resultDigit = digit1 + digit2 + mind;
			result += std::to_string(resultDigit % 10);
			mind = (resultDigit >= 10);
		}
	}
	else
	{
		for (int i = 0; i < HN1.mantissa.length(); i++)
		{
			int digit1 = HN1.mantissa[i] - '0';
			int digit2 = HN2.mantissa[i] - '0';
			int resultDigit = digit1 - digit2 - mind;
			result += std::to_string((resultDigit + 10) % 10);
			mind = (resultDigit < 0);
		}
	}
	result.insert(indexPoint, ".");
	std::reverse(result.begin(), result.end());
	HugeNumber Result(HN1.sign && HN2.sign, result, HN1.order);
	Result.RemoveExtraZeros();
	return Result;
}

HugeNumber operator/(const HugeNumber& hn, const int n)
{
	if (n == 0)
	{
		throw "Division by zero.";
	}
	HugeNumber HN(hn.sign, hn.mantissa + "00000000000000000000000000",
hn.order);
	HN.ChangeOrder();
	std::string ans;
	std::string number = HN.mantissa;
	const int indexPoint = HN.getIntPartLength() - (HN.mantissa[0] - '0' <
n);
	HN.mantissa.erase(HN.mantissa.find('.'), 1);
```

```
	int carry = 0;
	for (int i = 0; i < HN.mantissa.length(); i++)
	{
		long long cur = (HN.mantissa[i] - '0') + carry * 10;
		HN.mantissa[i] = int(cur / n) + '0';
		carry = int(cur % n);
	}
	HN.mantissa.insert(indexPoint, ".");
	if (HN.mantissa[0] == '.')
	{
		HN.mantissa = "0" + HN.mantissa;
	}
	HN.RemoveExtraZeros();
	return HugeNumber(HN.sign != (n < 0), HN.mantissa, HN.order);
}

void HugeNumber::NormalizeNumber()
{
	if (mantissa[0] == '0' && mantissa[1] == '.')
	{
		mantissa.erase(1, 1);
		int oldLength = mantissa.length();
		RemoveExtraZeros();
		int newLength = mantissa.length();
		int delta = oldLength - newLength;
		order -= delta;
		mantissa = "0." + mantissa;
		return;
	}
	int indexPoint = mantissa.find('.');
	mantissa.erase(indexPoint, 1);
	order += indexPoint;
	mantissa = "0." + mantissa;
	RemoveExtraZeros();
}

int HugeNumber::getIntPartLength()
{
	return mantissa.find('.');
}

int HugeNumber::getFracPartLength()
{
	return mantissa.length() - getIntPartLength() - 1;
}
```

Листинг файла main.cpp

```
#include <iostream>
#include "HugeNumber.h"
using namespace std;

int main()
{
	setlocale(LC_ALL, "Russian");

	HugeNumber hn1;
	HugeNumber hn2;
	HugeNumber result;

	hn1.ReadHugeNumber();
	hn2.ReadHugeNumber();
```

```
	std::cout << "Большое число #1: " << hn1;
	hn1.ChangeOrder();
	std::cout << " = " << hn1;
	hn1.NormalizeNumber();
	std::cout << " = " << hn1 << std::endl;

	std::cout << "Большое число #2: " << hn2;
	hn2.ChangeOrder();
	std::cout << " = " << hn2;
	hn2.NormalizeNumber();
	std::cout << " = " << hn2 << std::endl;

	result = hn1 + hn2;
	std::cout << "Сумма: " << result;
	result.ChangeOrder();
	std::cout << " = " << result;
	result.NormalizeNumber();
	std::cout << " = " << result << std::endl;

	result = result / 2;
	std::cout << "Среднее арифметическое: " << result;
	result.ChangeOrder();
	std::cout << " = " << result;
	result.NormalizeNumber();
	std::cout << " = " << result << std::endl;

	return 0;
}
```

Результат выполнения программы для задания

```
Введите число: 1e5
Введите число: 200e3
Большое число #1: 1.0e5 = 100000.0 = 0.1e6
Большое число #2: 200.0e3 = 200000.0 = 0.2e6
Сумма: 300000.0 = 300000.0 = 0.3e6
Среднее арифметическое: 150000.0 = 150000.0 = 0.15e6

Введите число: 152565326.4351e7
Введите число: 454652.52435e9
Большое число #1: 152565326.4351e7 = 1525653264351000.0 = 0.1525653264351e16
Большое число #2: 454652.52435e9 = 454652524350000.0 = 0.45465252435e15
Сумма: 1980305788701000.0 = 1980305788701000.0 = 0.1980305788701e16
Среднее арифметическое: 99015289435050.0 = 99015289435050.0 =
0.9901528943505e14

Введите число: 53426.345e-3
Введите число: 16.6342e2
Большое число #1: 53426.345e-3 = 53.426345 = 0.53426345e2
Большое число #2: 16.6342e2 = 1663.42 = 0.166342e4
Сумма: 1716.846345 = 1716.846345 = 0.1716846345e4
```

```
Среднее арифметическое: 85.8423172 = 85.8423172 = 0.858423172e2

Введите число: 44564564.656778687897987897656554e10
Введите число: 324345456465676768789.989786765645434423e15
Большое число #1: 44564564.656778687897987897656554e10 =
445645646567786878.97987897656554 = 0.44564564656778687897987897656554e18
Большое число #2: 324345456465676768789.989786765645434423e15 =
324345456465676768789989786765645434.423 =
0.32434545646567676878998978676564543442 3e36
Сумма: 324345456465676769235635433333432313.40287897656554 =
324345456465676769235635433333432313.40287897656554 =
0.32434545646567676923563543333343231340287897656554e36
Среднее арифметическое: 16217272823283838461781771666671615670143948828277 =
16217272823283838461781771666671615670143948828277 =
0.162172728232838384617817716666716156701439488282 77e36

Введите число: 5342534e3
Введите число: -76473e5
Большое число #1: 5342534.0e3 = 5342534000.0 = 0.5342534e10
Большое число #2: -76473.0e5 = -7647300000.0 = -0.76473e10
Сумма: -2304766000.0 = -2304766000.0 = -0.2304766e10
Среднее арифметическое: -1152383000.0 = -1152383000.0 = -0.1152383e10
```

Выводы:

В ходе работы были сформированы практические навыки моделирования операций над длинными числами.