МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Кафедра информационных технологий**

**КУРСОВАЯ РАБОТА**

**РАСШИРЕНИЕ БИБЛИОТЕКИ INFINITELY ДЛЯ ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ БАЗОВЫХ ОПЕРАЦИЙ НАД ЧИСЛАМИ C ФИКСИРОВАННОЙ ТОЧКОЙ И ПРОИЗВОЛЬНОЙ ТОЧНОСТЬЮ**

Работу выполнил\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_И.А.Бастрыкин

(подпись, дата) (инициалы, фамилия)

Факультет компьютерных технологий и прикладной математики 3 курс

Направление 02.03.03 – «Математическое обеспечение и администрирование информационных систем»

Научный руководитель, доц.

канд. физ.-мат. наук\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ю.В.Кольцов

(подпись, дата) (инициалы, фамилия)

Нормоконтролер, ст.преп.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.В.Харченко

(подпись, дата) (инициалы, фамилия)

Краснодар 2017

СОДЕРЖАНИЕ

Введение 4

1 Параллельная парадигма программирования 7

2 Обзор языка C# 11

2.1 История развития C# 11

2.2 Обзор .NET Framework 12

2.3 Структура BigInteger 13

2.3.1 Конструктор BigInteger 15

2.3.2 Свойства 15

2.4 Асинхронные вычислительные операции в C# 15

2.4.1 QueueUserWorkItem 15

2.4.2 Task 15

3 Программная реализация 16

3.1 Представление чисел в классе Infinitely 16

3.2 Конструктор класса Infinitely 16

3.3 Методы класса Infinitely 16

3.3.1 Сложение по модулю 17

3.3.2 Вычитание по модулю 17

3.3.3 Округление 17

3.3.4 Отбрасывание дробной части 17

3.4 Перегруженные операторы для класса Infinitely 16

3.4.1 Сложение 17

3.4.2 Вычитание 18

3.4.3 Умножение 19

3.4.4 Деление 19

3.4.5 Операции сравнения 19

3.4.6 Инкремент 19

3.4.6 Декремент 19

3.5 Асинхронное вычисления в классе Infinitely 22

3.4.1 Сложение 17

3.4.2 Вычитание 18

3.4.3 Умножение 19

3.4.3 Деление 19

Заключение 23

Список использованных источников 24

ВВЕДЕНИЕ

Известно, что компьютер может оперировать числами, количество бит которых ограниченно. Когда-то использовались 8-разрядные процессоры, затем – 16-, 32-, 64-разрядные. Как правило, мы привыкли работать с 32-х и 64-х разрядными целыми числами, которым на платформе .NET соответствуют типы Int32 (int) и Int64 (long) соответственно. Для бытовых целей чисел такой разрядности достаточно – как для представления целых чисел, так и для вещественных вычислений. Однако существуют приложения, где требуются вычисления с большими числами (например, криптография, астрономия) или нужно выполнять точные вычисления (например, в финансовой области), или же вычисления с произвольно заданной точностью.

Такие вычисления невозможно выполнить напрямую, поэтому в мире создано достаточно большое количество библиотек, облегчающих труд программистов в этой области. Как правило, большие числа представляются в этих библиотеках в виде структур.

Но для решения некоторых задач необходимо работать с числами, которые не вмещаются в эти типы данных. Именно для работы с такими большими числами существует длинная арифметика, позволяющая совершать операции над числами, разрядность которых превышает длину машинного слова данной вычислительной машины. Эти операции реализуются не аппаратно, а программно, используя базовые аппаратные средства работы с числами меньших порядков.

С момента появления многоядерных процессоров, повышение производительности вычислительных систем осуществляется в основном путем внедрения элементов параллелизма в компьютерных комплексах. Существующая сейчас массовая парадигма программирования — создание последовательных программ — стала противоречивой с появлением нескольких исполнителей в вычислительной системе.

Следуя современному направлению развития компьютерных технологий, было решено сделать библиотеку асинхронной.

Представленная курсовая работа посвящена реализации асинхронной библиотекидля арифметических операция с числами с фиксированной точкой и произвольной точностью.

Первая глава курсовой работы содержит обзор истории развития C# и его применения.

Вторая глава курсовой работы посвящена обзору возможностей языка C# для работы с длинными числами и асинхронными вычислениями.

Третья глава посвящена описанию процесса разработки библиотеки Infinitely.

1 Параллельная парадигма программирования

При решении задачи на параллельной вычислительной системе (несколько процессоров и/или несколько ядер) необходимо пройти совокупность восьми этапов:

1) Постановка задачи.

2) Создание математической модели.

3) Разработка алгоритма.

4) Декомпозиция алгоритма (decomposition).

При параллельной реализации алгоритма мы предполагаем, что он будет выполнен несколькими исполнителями. Для этого нужно декомпозировать алгоритм, то есть выделить в алгоритме наборы действий, которые могут быть исполнены одновременно, независимо друг от друга. Различают два вида декомпозиции — по данным и по вычислениям. Если в алгоритме сходным образом обрабатываются большие объемы данных, то можно попробовать разделить эти данные на части — зоны ответственности, каждая из которых допускает независимую обработку отдельным исполнителем, и выявить вычисления, связанные с зонами ответственности. Это — декомпозиция по данным.

Другой подход подразумевает разделение вычислений на зоны ответственности для их выполнения на разных исполнителях и определение данных, связанных с этими вычислениями. Это — декомпозиция по вычислениям (функциональная декомпозиция).

Декомпозиция не всегда бывает возможна. Существуют алгоритмы, которые никак не допускают при своей реализации участия нескольких исполнителей.

5) Назначение работ (assignment).

После успешного завершения этапа декомпозиции весь алгоритм представляет собой совокупность множеств наборов действий, направленных на решение подзадач отдельными исполнителями. Наборы действий одного множества допускают одновременное и независимое выполнение. Множества могут содержать разное количество наборов и реализовываться на различном количестве исполнителей.

Число имеющихся ядер всегда ограничено. На данном этапе необходимо определить, сколько исполнителей будет задействовано и как по ним распределить подзадачи. Главными целями назначения подзадач являются балансировка загрузки процессоров (ядер), уменьшение обменов данными между ними и уменьшение накладных расходов на выполнение самого назначения. Способы назначения разделяются на две категории по времени:

статические — распределение выполняется на этапе написания, компиляции или старта программы (до реального начала вычислений);

динамические — распределение осуществляется в процессе исполнения программы.

6) Дирижирование (orchestration).

Целью данного этапа является выбор программной модели и определение требуемой синхронизации работы исполнителей, которая будет во многом зависеть от программной модели.

Классификация и названия программных моделей до конца не устоялись, но можно выделить четыре основных модели:

— Последовательная модель. Полагает, что пишется обычная последовательная программа в одной из последовательных моделей программирования для последующего автоматического ее распараллеливания компилятором или специальными программными средствами. Преимущество модели — не надо делать ничего лишнего по сравнению с последовательным вариантом, недостаток — автоматическое распараллеливание имеет крайне ограниченные возможности.

— Модель передачи сообщений. Полагает, что работающее приложение состоит из набора процессов с различными адресными пространствами, каждый из которых работает на своем исполнителе. Процессы обмениваются данными с помощью передачи сообщений через явные операции send/receive. Преимущество данной модели заключается в том, что программист осуществляет полный контроль над решением задачи, недостаток — в сложности программирования.

— Модель разделяемой памяти. Полагает, что приложение состоит из набора нитей исполнения (thread’ов), использующих разделяемые переменные и примитивы синхронизации. Выделяются две подмодели: явные нити исполнения — использование системных или библиотечных вызовов для организации работы thread’ов и программирование на языке высокого уровня с использованием соответствующих программ.

Первая подмодель обладает хорошей переносимостью, даёт полный контроль над выполнением, но очень трудоемка. Вторая подмодель легка для программирования, но не даёт возможности полностью контролировать решение задачи.

— Модель разделенных данных. Полагает, что приложение состоит из наборов процессов или thread’ов, каждый из которых работает со своим набором данных, обмена информацией при работе нет. Применима к ограниченному классу задач.

7) Написание программы, реализующей алгоритм, в выбранной модели программирования и на выбранном алгоритмическом языке.

8) Отображение (mapping). При запуске программы на параллельной компьютерной системе необходимо сопоставить виртуальным исполнителям, появившимся на предыдущих этапах парадигмы программирования, реальные физические устройства. В зависимости от выбранной модели программирования это может осуществляться как лицом, проводящим вычислительный эксперимент, так и операционной системой.

2 Обзор языка C#

C# — язык программирования, сочетающий объектно-ориентированные и контекстно-ориентированные концепции. Разработан в 1998—2001 годах компанией Microsoft как основной язык разработки программ для платформы .NET Framework корпорации Microsoft. Компилятор с C# входит в стандартную установку самой .NET, поэтому программы на нём можно создавать и компилировать без инструментальных средств.

Язык имеет строгую статическую типизацию, поддерживает полиморфизм, перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения, комментарии в формате XML. В C# удачно сочетаются испытанные средства программирования с самыми последними новшествами и предоставляется возможность для эффективного и очень практичного написания программ, предназначенных для вычислительной среды современных предприятий.

2.1 История развития языка C#

В таблице 1 представлена история развития языка C#.

Таблица 1 — История развития языка C#

|  |  |  |
| --- | --- | --- |
| Год, месяц | Версия языка | Примечание |
| 2002, Январь | **C# 1.0** | Первая версия языка |
| 2005, Сентябрь | **C# 2.0** |  |
| 2007, Август | **C# 3.0** |  |
| 2010, Апрель | **C# 4.0** |  |
| 2013, Июнь | **C# 5.0** | Асинхронные методы  Сведения о вызывающем объекте |
| 2015, Июль | **C# 6.0** |  |

2.1 Обзор .NET Framework

Программа на языке C# выполняется в среде .NET Framework – интегрированном компоненте Windows, содержащем виртуальную систему выполнения (среда CLR) и унифицированный набор библиотек классов. Среда CLR представляет собой коммерческую реализацию Майкрософт инфраструктуры CLI (common language infrastructure), международного стандарта, основы сред выполнения и разработки с тесным взаимодействием языков и библиотек.

Исходный код, написанный на языке C#, компилируется в промежуточный язык (IL) в соответствии со спецификацией CLI. Код IL и ресурсы, такие как растровые изображения и строки, хранятся на диске в исполняемом файле, называемом сборкой, с расширением EXE или DLL в большинстве случаев. Сборка содержит манифест со сведениями о типах сборки, версии, языке и региональных параметрах и требованиях безопасности.

Важным свойством платформы является межъязыковое взаимодействие. Это возможность кода, написанного на одном языке, без труда взаимодействовать с кодом, написанным на другом языке. Межъязыковая возможность взаимодействия требуется для построения крупных, распределённых программных систем. Также она желательна для создания отдельных компонентов программ, так как наиболее ценным компонентом считается тот, который может быть использован в разных языках программирования и в большем числе операционных сред.

Также важным свойством является полная интеграция с платформой Windows.

Назначение .NET Framework – служить средой для поддержки разработки и выполнения сильно распределённых компонентных приложений. Она обеспечивает совместное использование разных языков программирования, а также безопасность, переносимость программ и общую модель программирования для платформы Windows.

2.3 Структура BigInteger

Платформа .NET вплоть до 4.0 версии не имела встроенной поддержки работы с длинными числами. В четвёртой версии .NET обрела функционал, который доступен через сборку System.Numerics и тип BigInteger определенные в одноимённом с названием сборки пространстве имён.

BigInteger позволяет работать с произвольно большими целыми числами со знаком.

2.3.1 Конструктор [BigInteger(<value>)](https://msdn.microsoft.com/ru-ru/library/dd268207(v=vs.110).aspx)

Конструктор [BigInteger(<value>)](https://msdn.microsoft.com/ru-ru/library/dd268207(v=vs.110).aspx) инициализирует новый экземпляр BigInteger структуры с помощью значений в массиве байтов (Byte[]), Decimal значение, значений с плавающей запятой двойной точности (Double), 32-разрядного знакового целочисленного значения (Int32), 64-разрядного знакового целочисленное значение (Int64), значение с плавающей запятой одиночной точности (Single), 32-разрядное целочисленное значение без знака (UInt32), 64-разрядное целочисленное значение без знака (UInt64).

2.3.2 Свойства

isEven - определяет является ли значение текущего BigInteger объекта четным числом.

[IsOne](https://msdn.microsoft.com/ru-ru/library/system.numerics.biginteger.isone(v=vs.110).aspx) - определяет является ли значение текущего BigInteger объекта единицей

IsPowerOfTwo - Указывает, является ли значение текущего BigInteger объекта представляет собой степень числа два.

IsZero - Указывает, является ли значение текущего BigInteger объект BigInteger.Zero.

MinusOne - Получает значение, представляющее минус единицу (-1).

One - Получает значение, представляющее единицу (1).

Sign - Возвращает число, указывающее знак (минус, плюс или нуль) текущего BigInteger объекта.

Zero - Получает значение, представляющее 0 (ноль).

В общем, структура BigInteger является полноценной реализацией длинной арифметики на платформе .NET. При этом Microsoft постаралась максимально близко приблизить её к примитивным числовым типам: экземпляр BigInteger можно использовать точно так же, как и любой другой целочисленный тип. BigInteger перегружает стандартные числовые операторы для выполнения основных математических операций, таких как сложение, вычитание, деление, умножение, вычитания, отрицание. Можно также использовать стандартные числовые операторы для сравнения двух значений BigInteger друг с другом. Как и другие типы целого числа, BigInteger поддерживает битовые операторы And, Or, XOR, сдвиг влево и сдвиг вправо.  
Для языков, не поддерживающих пользовательские операторы, структура BigInteger также предоставляет эквивалентные методы для выполнения математических операций. Это относится к методам Add, Divide, Multiply, Negate, Subtract и некоторым другим. Точно так же Microsoft поступило в реализации структуры Decimal.

2.4 Средства для асинхронных вычислений в языке C#

Создание и уничтожение потока занимает изрядное время. Кроме того, при наличии множества потоков впустую расходуется память и снижается производительность, так как операционной системе приходится планировать исполнение потоков и выполнять переключения контекста. Но среда CLR способна управлять собственным пулом потоков, то есть набором готовых потоков, доступных для использования приложениями. Для каждого экземпляра CLR существует свой пул, используемый всеми доменами приложений, находящимися под управлением экземпляра CLR. Если в один процесс загружаются несколько экземпляров CLR, для каждого из них формируется собственный пул.

При инициализации CLR пул потоков пуст. В его внутренней реализации поддерживается очередь запросов на выполнение операций. Для выполнения приложением асинхронной операции вызывается метод, размещающий соответствующий запрос в очереди пула потоков. Код пула извлекает записи из очереди и распределяет их между потоками из пула. Если пул пуст, создается новый поток. Создание потока негативно сказывается на производительности, но после завершения исполнения своего задания поток не уничтожается, а возвращается в пул и ожидает следующего запроса. Так как поток не уничтожается, производительность не падает.

Когда приложение отправляет пулу много запросов, он пытается обслужить их все с помощью одного потока. Однако если приложение создает очередь запросов быстрее, чем поток из пула их обслуживает, создаются дополнительные потоки. Такой подход позволяет обойтись при обработке запросов небольшим количеством потоков.

Когда приложение прекращает отправлять запросы в пул, появляются незанятые потоки, впустую занимающие память. Поэтому через некоторое время бездействия (различное для разных версий CLR) поток пробуждается самоуничтожается, освобождая ресурсы.

2.4.1 QueueUserWorkItem

Для добавления в очередь пула потоков асинхронных вычислительных операций обычно вызывают один из следующих методов класса ThreadPool:

static Boolean QueueUserWorkItem(WaitCallback callBack);

static Boolean QueueUserWorkItem(WaitCallback callBack, Object state);

Эти методы ставят «рабочий элемент» вместе с дополнительными данными состояния в очередь пула потоков и сразу возвращают управление приложению. Рабочим элементом называется указанный в параметре callback метод, который будет вызван потоком из пула. Этому методу можно передать один параметр через аргумент state (данные состояния). Без этого параметра версия метода QueueUserWorkItem передает методу обратного вызова значение null. Все заканчивается тем, что один из потоков пула обработает рабочий элемент, приводя к вызову указанного метода. Создаваемый метод обратного вызова должен соответствовать делегату System.Threading.WaitCallback, который определяется так:

delegate void WaitCallback(Object state);

Планировщик Windows решает, какой из потоков должен выполняться первым, или же планирует их одновременное выполнение на многопроцессорном компьютере.

2.4.2 Task

Вызвать метод QueueUserWorkItem класса ThreadPool для запуска асинхронных вычислительных операций очень просто. Однако этот подход имеет множество недостатков. Самой большой проблемой является отсутствие встроенного механизма, позволяющего узнать о завершении операции и получить возвращаемое значение. Для обхода этих и других ограничений специалисты Microsoft ввели понятие заданий(tasks), выполнение которых осуществляется посредством типов из пространства имен System.Threading.Tasks.

Для создания объекта Task следует вызвать конструктор и передать ему делегата Action или Action<Object>, указывающего, какую операцию вы хотите выполнить. При передаче метода, ожидающего тип Object, в конструктор объекта Task следует передать также аргумент, который должен быть в итоге передан операции. При вызове Run передается делегат Func<TResult> или Action, определяющий выполняемую операцию.

3 Программная реализация

В рамках курсовой работы была начата реализация асинхронной библиотеки Infinitely для арифметических операций с числами с фиксированной точкой и произвольной точностью. На данный момент реализация библиотеки предоставляет следующие функции операции сравнения (>, <, =), сложения, вычитания (а так же сложения и вычитания по модулю), умножения и деления.

При создании нового объекта класса Infinitely необходимо передать конструктору параметры, определяющие количество цифр в числе и количество знаков после запятой.

3.1 Представление чисел в классе Infinitely

В классе Infinitely числа представлены, как массив элементов типа long (Int64), т.е. как массив целых чисел, кодируемых 64 битами. Для кодировки чисел от 0 до 9 необходимо задействовать 4 бита. Каждый элемент массива делится на по 4 бита на 16 частей. Таким образом, каждый элемент массива способен хранить 16 цифр. Имея массив из N элементов мы можем хранить число с количество значащих цифр, равным N\*16.

Для определения фиксированной точки в классе Infinitely имеется поле fractionDigitsNumer, в которое хранит в себе количество цифр после запятой.

3.2 Конструктор Infinitely

Конструктор Infinitely принимает на вход два параметра и имеет вид Infinitely(long allDigitsNumber, long fractionDigitsNumber), где

allDigitsNumber – максимально возможное количество цифр числа, а

fractionDigitsNumber – количество цифр после запятой. Определяется массив number с числом (1 + allDigitsNumber / 2) переменных типа long. Поле attribute=0.

3.3 Методы Infinitely

Метод Infinitely toInfinitely(type value, Infinitely inf) – записывает в переменную inf типа Infinitely значение переменной value типа type с заданной ранее точностью, где type может быть int, long, decimal или string.

Метод Infinitely Copy(Infinitely inf) — возвращает копию экземпляра inf типа Infinitely.

Метод bool isZero(Infinitely inf) — проверяет, содержит ли экземпляр inf типа Infinitely значение, равное нулю и записывает в поле attribute значение 0.

Метод bool AbsGreater(Infinitely inf\_1, Infinitely inf\_2) — сравнивает по модулю числовые значения, содержащиеся в переменных inf\_1 и inf\_2. Возвращает true, если числовое значение inf\_1 больше по модулю числового значения inf\_2.

Метод Infinitely AbsPlus(Infinitely inf\_1, Infinitely inf\_2) — возвращает сумму абсолютных значений, содержащихся в inf\_1 и inf\_2.

Метод Infinitely AbsMinus(Infinitely inf\_1, Infinitely inf\_2) — возвращает абсолютную разность значений, содержащихся в inf\_1 и inf\_2.

Метод Show(Infinitely inf) — выводит на консоль числовое значение экземпляра inf.

Метод Round(Infinitely inf) — округляет числовое значение inf.

Метод Trunc(Infinitely inf) — отбрасывает дробную часть числового значения inf.

3.4 Перегруженные операторы для класса Infinitely

3.4.1 Сложение и вычитание

Сложение и вычитание производятся схожим образом и основываются на методах сложения и вычитания по модулю.

3.3.1.1 Сложение по модулю

Метод AbsPlus(Infinitely inf\_1, Infinitely inf\_2) возвращает экземпляр типа Infinitely, содержащий число, равное сумме модулей чисел inf\_1 и inf\_2.

Сложение осуществляется поразрядно. Для этого из каждого элемента массива number[] экземпляров inf\_1 и inf\_2, с помощью операций сдвига поочерёдно по 4 бита вычленяется каждая цифра числа inf\_1 и складывается с цифрой того же разряда числа inf\_2, а также с переменной mod, которая в начале операции сложения равна нулю и представляет собой выход за разряд от сложения предыдущих цифр. Результат записывается в переменную t типа long. Если полученный результат сложения двух цифр оказывается больше девяти (t > 9), то в соответствующий разряд результирующего числа inf\_3 записывается остаток от целочисленного деления переменной t на 10, а в переменную mod записывается единица (mod = 1). Если же результат сложения цифр меньше либо равен девяти (t<=9), то в соответствующий разряд результирующего числа inf\_3 записывается переменная t, а в переменную mod записывается ноль (mod = 0).

3.3.1.2 Вычитание по модулю

Метод AbsMinus(Infinitely inf\_1, Infinitely inf\_2) возвращает экземпляр типа Infinitely, содержащий число, равное разности модулей чисел inf\_1 и inf\_2.

Для начала выполняется сравнение по модулю чисел inf\_1 и inf\_2. Если inf\_1 оказывается меньше inf\_2, то вычитаться будет inf\_1 из inf\_2, иначе вычитаться будет inf\_2 из inf\_1. Таким образом, произойдёт вычитание меньшего из большего. Далее ход метода аналогичен методу сложения по модулю. Из каждого элемента массива number[] экземпляров inf\_1 и inf\_2, с помощью операций сдвига поочерёдно по 4 бита вычленяется каждая цифра числа inf\_1 и вычитается из неё вычитается цифра того же разряда числа inf\_2, а также значение переменной mod, которая в начале операции сложения равна нулю и представляет собой занимание разряда от для вычитания предыдущих цифр. Результат записывается в переменную t типа long. Если полученный результат сложения двух цифр оказывается меньше нуля (t < 0), то в соответствующий разряд результирующего числа inf\_3 записывается значение переменной t, увеличенной на 10, а в переменную mod записывается единица (mod = 1). Если же результат сложения цифр больше нуля (t>0), то в соответствующий разряд результирующего числа inf\_3 записывается переменная t, а в переменную mod записывается ноль (mod = 0).

3.3.1.3 Сложение

Сначала проверяется значение полей attribute экземпляров inf\_1 и inf\_2. Если у одного из слагаемых поле attribute равно нулю, то возвращаемым результатом будет другое слагаемое. Если слагаемые имеют одинаковый знак, то происходит сложение по модулю чисел из inf\_1 и inf\_2, после выполнения которого, полю attribute результирующего значения присваивается содержимое поля attribute экземпляра inf\_1. Если слагаемые имеют различные значения поля attribute (то есть имеют различные знаки), то производится вычитание по модулю, после чего, полю attribute результирующего значения присваивается содержимое поля attribute большего по модулю из элементов inf\_1 и inf\_2.

3.3.1.4 Вычитание

Сначала проверяется значение полей attribute экземпляров inf\_1 и inf\_2. Если у первого операнда поле attribute равно нулю, то возвращается второй операнд с противоположным знаком. Если у второго операнда поле attribute равно нулю, то возвращается первый операнд. Если операнды имеют одинаковое значение поля attribute (то есть имеют одинаковый знак), то происходит сложение по модулю чисел из inf\_1 и inf\_2, после выполнения которого, полю attribute результирующего значения присваивается содержимое поля attribute экземпляра inf\_1. Если операнды имеют различные значения поля attribute (то есть имеют различные знаки), то производится вычитание по модулю, после чего, полю attribute результирующего значения присваивается содержимое поля attribute большего по модулю из элементов inf\_1 и inf\_2.

3.3.2 Умножение

Возвращает экземпляр типа Infinitely, содержащий число, равное произведению чисел inf\_1 и inf\_2. Сначала проверяется значение полей attribute экземпляров inf\_1 и inf\_2. Если хотя бы у одного из операндов поле attribute равно нулю, то возвращаемым значением будет ноль, иначе значения attribute перемножаются и записываются в соответствующее поле результирующего объекта inf\_3.

Умножение осуществляется поразрядно. Для этого из каждого элемента массива number[] экземпляров inf\_1 и inf\_2, с помощью операций сдвига поочерёдно по 4 бита вычленяется каждая цифра числа inf\_1, умножается на каждую цифру числа inf\_2 и складывается с цифрой соответствующего разряда числа inf\_3, а также с переменной mod, которая в начале операции сложения равна нулю и представляет собой выход за разряд от умножения на предыдущую цифру. Результат записывается в переменную t типа long. Если полученный результат сложения двух цифр оказывается больше девяти (t > 9), то в соответствующий разряд результирующего числа inf\_3 записывается остаток от целочисленного деления переменной t на 10, а в переменную mod записывается целая часть от деления t на 10 (mod = t / 10). Если же результат сложения цифр меньше либо равен девяти (t<=9), то в соответствующий разряд результирующего числа inf\_3 записывается переменная t, а в переменную mod записывается ноль (mod = 0).

3.3.3 Деление

Возвращает экземпляр типа Infinitely, содержащий число, равное частному чисел inf\_1 и inf\_2. Сначала проверяется значение полей attribute экземпляров inf\_1 и inf\_2. Если у делимого объекта поле attribute равно нулю, то возвращаемым значением будет ноль. Иначе значения attribute делятся друг на друга и записываются в соответствующее поле результирующего объекта inf\_3. В случае, когда attribut делимого объекта равен нулю, срабатывает исключение DivideByZeroException.

Деление осуществляется поразрядно. Для этого из каждого элемента массива number[] экземпляров inf\_1 и inf\_2, с помощью операций сдвига поочерёдно по 4 бита вычленяется каждая цифра. Действие алгоритма начинается в старшем разряде делимого, то есть inf\_1. Если это возможно, происходит вычитание делителя, то есть inf\_2, из старших разрядом делимого. Как только пропадает возможность проводить такое вычитание без остатка, количество проведённых делений записывается в соответствующий разряд результирующего объекта и процесс повторяется для следующего разряда делимого.

3.3.4 Операции сравнения

А) Операторы >, <, >=, <= возвращают в качестве результата значение логического типа bool. В случае с операторами <, > сначала проверяются знаки чисел. Если знаки равны, находится количество разрядов от

3.5 Асинхронные выполнение умножения в классе Infinitely

3.5.1 Сложение

<тесты>

3.5.2 Вычитание

<тесты>

3.5.3 Умножение

А) Умножение в лоб.

Метод AsyncMull(Infinitely inf\_1, Infinitely inf\_2) возвращает результат асинхронного перемножение чисел, хранящихся в inf\_1 и inf\_2.

Для каждой цифры числа inf\_1 вызывается асинхронный метод Mull(Object state), умножающий эту цифру на каждую цифру числа, хранящегося в inf\_2, и поразрядно прибавляющий получившиеся числа к результату.

Для передачи значений была описана структура Point, содержащая индекс цифры в массиве number переменной типа Infinitely, а также ссылки на множитель, множимое и результат типа Infinitely.

Перед подачей переменной p типа Point в метод Mull, в её поля записываются значения индексов и ссылок. Затем, уже в самом методе Mull, эти значения извлекаются из переменной state типа Object.

Б) Умножение методом Карацубы.

Метод умножения Карацубы относится к парадигме «[разделяй и властвуй](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D1%8F%D0%B9_%D0%B8_%D0%B2%D0%BB%D0%B0%D1%81%D1%82%D0%B2%D1%83%D0%B9_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))». [Сложность вычисления](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_(%D0%B1%D0%B8%D1%82%D0%BE%D0%B2%D0%B0%D1%8F)) алгоритма nlog23

Данный алгоритм представляет собой простую реализацию идеи разделения входных данных. Идея заключается в разделении одной операции умножения над N-{\displaystyle n}значными числами на три операции умножения над числами длины N/2{\displaystyle n/2}N/N{\displaystyle O(n)}.

Для перемножения двух чисел, превышающих длину машинного слова, алгоритм Карацубы вызывается рекурсивно до тех пор, пока эти числа не станут достаточно маленькими, чтобы их можно было перемножить непосредственно.

<тесты>

3.5.4 Деление

<тесты>

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы было проведено расширение функционала асинхронной библиотеки Infinitely для повышения эффективности базовых операций над числами с фиксированной точкой и произвольной точностью. Пока что в библиотеке Infinitely асинхронным является только умножение, но в будущем планируется расширение функционала и использования асинхронных вычислений. Так же планируется создание автоматизированных тестов для проверки точности результатов описанных функций. Когда данная библиотека будет завершена, она сможет поспособствовать повышению точности вычислений, что поможет получить более точные результаты, и тем самым поспособствовать решению задач в самых различных областях.

СПИСОК ИСТОЧНИКОВ

1. Карпов В. Е. Введение в распараллеливание алгоритмов и программ: Московский физико-технический институт, 2010.
2. Рихтер Дж. CLR via C# Программирование на платформе Microsdoft .NET Framework 4.5 на языке C#.
3. Структура BigInteger – MSDN. URL: <https://msdn.microsoft.com/ruru/library/system.numerics.biginteger(v=vs.110).aspx>, дата обращения: 02.12.2016.