

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА
на тему
ПРОГРАММНЫЙ МОДУЛЬ
НЕПОЗИЦИОННОЙ ОБРАБОТКИ
ЦИФРОВЫХ ДАННЫХ

Студент


(подпись, дата)

Ванюшин Иван Алексеевич
(ФИО полностью)

Руководитель


(подпись, дата)

Егоров В.Ю.
(фамилия, инициалы)

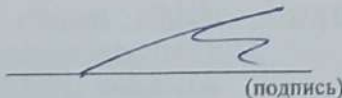
Нормоконтролёр


(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от
20 июня 2025г., № 43)

Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

«6» мая 2025 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Ванюшин Иван Алексеевич гр. 21ВВП2 факультета ФВТ

2. Тема работы Программный модуль «RNSlib». Программный модуль непозиционной обработки цифровых данных

Тема утверждена приказом ПГУ № 601/о от "6" мая 2025 г.

3. Руководитель бакалаврской работы Егоров В.Ю.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать программное обеспечение для обработки данных в системе остаточных классов. Тип обрабатываемых данных: целочисленный беззнаковый формат, разрядностью 8, 16, 32 разряда, целочисленный знаковый формат разрядностью 8, 16, 32 разряда, формат с фиксированной точкой разрядностью 16, 32 разряда. Программное обеспечение реализует следующий набор арифметических операций: сложение, вычитание, умножение, деление. Программное обеспечение должно быть реализовано с помощью следующего стека технологий: язык программирования C/C++, IDE Visual Studio 2019, инструменты сборки CMake. Программный модуль должен иметь возможность встраивания в зависимые проекты.

5. Перечень подлежащих разработке вопросов

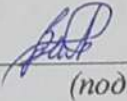
Требуется произвести анализ предметных областей: изучить существующие средства непозиционной арифметики, выделить их достоинства и недостатки. Изучить функционал библиотек для использования непозиционной арифметики. Изучить платформы, в которые можно имплементировать программный модуль. Выбрать язык программирования, среду разработки, средства построения проекта. Определить архитектуру программного обеспечения.

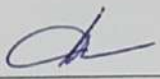
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	26.04.2025 – 28.05.2025	
2	Анализ требований	32	29.04.2025 – 30.05.2025	
3	Проектирование диаграмм отношений классов	30	31.04.2025 – 3.05.2025	
4	Выбор технологий, применяемых при проектировании и разработке	40	4.05.2025 – 8.05.2025	
5	Разработка основной части программы	70	16.05.2025 – 23.05.2025	
6	Оформление пояснительной записки	54	24.05.2025 – 31.05.2025	

Дата выдачи задания "26" мая 2025 г.

Руководитель бакалаврской работы  06.05.2025 Егоров В.Ю.
(подпись, дата)

Задание к исполнению принял студент  06.05.2025 Ванюшин И.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  19.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 73 л., 1 рис., 1 таблица, 8 источн. и 3 прил.

НЕПОЗИЦИОННАЯ АРИФМЕТИКА, МОДУЛЬНЫЕ КОМБИНАЦИИ, ГОМОМОРФНЫЕ ВЫЧИСЛЕНИЯ, КРОССПЛАТФОРМЕННОСТЬ, C/C++, CMAKE

Цель работы – разработка программного обеспечения, предоставляющего функционал для работы с числами в системе остаточных классов с целью повышения производительности операций над большими числами и шифрования облачных вычислений.

Для достижения поставленной цели разработана библиотека, отвечающая задачам, поставленным в техническом задании: возможность обработки целых чисел со знаком, возможность обработки целых чисел без знака, возможность обработки чисел с фиксированной точкой, реализация арифметических операций сложения, вычитания, умножения, деления в системе остаточных классов, , предоставление интуитивно понятного интерфейса для вызова функций разработанного программного обеспечения.

В результате проведенной работы разработан кроссплатформенное программное обеспечение для работы в системе остаточных классов на языке C/C++.

					ПГУ 1.090301.03.001 ПЗ			
Изм	Лист	№ докум.	Подпись	Дата	Программный модуль непозиционной обработки цифровых данных Пояснительная записка	Литера	Лист	Листов
Разраб.		Ванюшин И.А.		18.06.25				
Пров.		Евгоров В.Ю.		18.06.25			1	73
Н. Контр.		Бычков А.С.		18.06.25		ФВТ, каф. ВТ гр. 21ВВП2		
Утв.		Митрохин М.А.		18.06.25				

Содержание

Введение.....	6
1 Программный модуль непозиционной обработки цифровых данных	8
1.1 Анализ предметной области	8
1.1.1 Актуальность разработки.....	8
1.1.2 Анализ существующих аналогов	10
1.1.3 Обоснование актуальности разработки программы.....	12
1.1.4 Постановка задачи	12
1.2 Выбор технологий и средств разработки приложения	14
1.2.1 Выбор языка программирования.....	14
1.2.2 Выбор средств построения проекта	16
1.2.3 Выбор среды разработки.....	17
1.3 Проектирование программного модуля.....	17
1.4 Разработка программного модуля	19
1.4.1 Математическое обоснование	19
1.4.1.1 Целочисленный тип без знака	19
1.4.1.2 Приведение к типу формата СОК	19
1.4.1.3 Реализация арифметических операций.....	20
1.4.1.4 Приведение к простейшему типу	22
1.4.1.5 Целочисленный тип со знаком	23
1.4.1.6 Тип фиксированной точки	25
1.4.1.7 Нормализация по модулю 256.....	26
1.4.2 Описание программного модуля.....	30
1.4.4 Тестирование и отладка.....	32
2 Охрана труда.....	33

Заключение	36
Список использованных источников	38
Приложение А – UML-диаграммы	39
Приложение Б – листинги программы.....	45
Приложение В – презентация.....	67

					ПГУ 1.090301.03.001 ПЗ	Лист
						4
Изм.	Лист	№ докум.	Подпись	Дата		

Перечень сокращений и обозначений

В настоящей работе применяются следующие обозначения и сокращения

СОК – система остаточных классов

ПСС – позиционная система счисления

НСС – непозиционная система счисления

ЭВМ – электронно-вычислительная машина

BGV (Brakerski-Gentry-Vaikuntanathan scheme) – схема гомоморфного вычисления второго поколения

CKKS (Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song scheme) – схема гомоморфного вычисления четвёртого поколения

КТО – китайская теорема об остатках

					ПГУ 1.090301.03.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В век стремительного развития информационных технологий почти у каждого человека есть выход в глобальную информационную сеть. Портативные и настольные вычислительные устройства стали неотъемлемой частью нашей жизни, предоставляя быстрый доступ к необходимой информации и ее хранению, а большая часть бумажной волокиты стала набором файлов в папке облачного или локального хранилища данных. Однако с этими преимуществами приходят и недостатки: архитектура большинства стандартных вычислительных машин нацелена на работу со строками, и имеет ограниченный набор базовых типов данных, а облачные вычисления не всегда защищают пользователя от кражи его данных злоумышленником.

Современные алгоритмы шифрования позволяют организовать стабильные и защищенные каналы для передачи данных, однако вместе с этим имеют дополнительные ограничения в контексте преобразования данных. Такими ограничениями будут считаться программные затраты на дешифрацию данных для их преобразования и необходимость шифрования результата их обработки.

Для обхода таких ограничений существует специальный непозиционный формат числовых данных, именуемый системой остаточных классов, где каждое число представлено набором его остатков от деления на основания, являющиеся набором простых чисел. Такая форма записи чисел позволяет хранить числовые данные, а также выполнять над ними простые арифметические операции без нужды в обратной дешифрации информации. Помимо этого, разряды числа в формате СОК являются независимыми, из-за чего арифметические действия на каждом отдельном разряде могут выполняться параллельно за одно и то же время вне зависимости от величины числа. Все эти факторы позволяют использовать числа непозиционной системы счисления как основу для гомоморфной арифметики в контексте защищенных вычислений, а также как тип данных для организации арифметики над большими числами.

Возникает потребность в разработке программного модуля, способного перекрыть все недостатки работы со сложными числами в стандартной архитектуре вычислительных систем, а также способного перекрыть ограничения, накладываемые традиционными алгоритмами шифрования. Требуется создать такую прикладную библиотеку, которая предоставит инструменты для шифростойких вычислений на удалённых рабочих станциях и методы для работы с большими числами. Именно поэтому тема «Программный модуль непозиционной обработки цифровых данных» в качестве выпускной квалификационной работы является чрезвычайно актуальной.

Целью выпускной работы является:

- исследование предметной области и анализ существующих решений;
- разработка прикладного кроссплатформенного модуля для производства гомоморфных вычислений и вычислений с числами больших размерностей;
- проектирование и разработка программного интерфейса;
- тестирование разработанной системы;
- внедрение модуля непозиционной обработки цифровых данных.

Проект станет основой для производства защищенной обработки цифровых данных на удалённых сервисах, а также позволит работать с арифметикой чисел больших размерностей.

1 Программный модуль непозиционной обработки цифровых данных

1.1 Анализ предметной области

1.1.1 Актуальность разработки

Современные информационные технологии имеют тенденцию к повышению объемов передаваемых данных, работе с большими данными и повышенному вниманию к вопросам информационной безопасности. Во многих случаях сервисам передаются на обработку конфиденциальные данные клиентов без их возможности полностью проконтролировать то, как эти данные будут использованы.

Одним из направлений для решения этой проблемы является использование гомоморфных вычислений. Гомоморфные вычисления позволяют выполнять простые арифметические действия над данными без необходимости их дешифрования. Другими словами, приходящие от клиента зашифрованные данные не требуется расшифровывать, чтобы произвести вычисление результата, а сам результат не требуется шифровать. Такая технология позволит по-новому взглянуть на передачу конфиденциальной информации, например, в сферах здравоохранения, в сфере финансов или даже в сферах юридической или государственной документации.

В сфере медицины – это может стать инструментом для сбора и анализа статистики без раскрытия личных данных пациентов. В секторе экономики – это может помочь в проведении защищенных финансовых транзакций и их анализа без раскрытия конкретных данных в виде фактических сумм или субъектов транзакций. В государственном секторе – это будет важным инструментом при работе с защищенными данными, вроде налоговых отчетов, данных переписей населения или данных голосований.

Стандартные позиционные системы счисления в рамках современных вычислительных систем менее приспособлены для произведения защищенных вычислений. Одним из факторов, указывающих на это, являются зависимость

					ПГУ 1.090301.03.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

разрядов числа от более младших разрядов или, так называемый, перенос (carry), возникающий при выполнении базовых арифметических операций. Перенос в рамках этой системы ограничивает нас от использования параллельных вычислений, а также повышает вероятность хакерских атак по побочным каналам, основой которых служит анализ времени выполнения команд, затрат электроэнергии или особенностей обращения к памяти.

В контексте рассматриваемого вопроса особый интерес предоставляют непозиционные системы счисления, в частности, система остаточных классов. Главное отличие системы остаточных классов от обычных позиционных систем счисления состоит в том, что разряды чисел такого формата являются независимыми друг от друга. Это позволяет производить полностью независимые и параллельные вычисления за одинаковое время вне зависимости от величины чисел, что также делает систему менее уязвимой к атакам по побочным каналам в том числе.

Несмотря на явные преимущества использования чисел системы остаточных классов в рамках шифростойких вычислений, на данный момент существует не так много универсальных прикладных инструментов, позволяющих производить подобные вычисления. Большинство из таких модулей либо имплементированы в прикладные криптографические библиотеки, либо не имеют удобного интерфейса для работы с разными типами чисел в формате системы остаточных классов, в том числе с числами с фиксированной точкой.

Таким образом, проектирование и написание прикладного модуля для выполнения арифметических действий над числами непозиционной системы счисления является значимым не только с технической стороны вопроса, но и станет основой для построения более сложных модулей гомоморфных вычислений для выполнения защищенных преобразований данных на различных цифровых платформах с высоким требованием к безопасности данных.

1.1.2 Анализ существующих аналогов

Среди косвенных аналогов модуля непозиционной обработки цифровых данных можно выделить следующие решения:

- модуль гомоморфного шифрования «Microsoft SEAL» (C++/C#) [1];
- модуль гомоморфного шифрования «HElib» (C++) [2];
- математический модуль «NTL» (C++) [3].

Упомянутые прикладные модули имеют открытый исходный код за авторством компаний IBM и Microsoft за исключением модуля «NTL».

Рассмотрим инструменты, предоставляемые модулем гомоморфного шифрования «Microsoft SEAL». Данный прикладной модуль предоставляет методы для шифрования целых и реальных чисел по схемам BGV и CKKS, а также базовые арифметические методы, такие как сложение и умножение упомянутых типов чисел в непозиционной системе счисления.

Данный модуль обладает следующими преимуществами:

- кроссплатформенность;
- высокий уровень оптимизации при работе с различными схемами шифрования;
- поддержка работы чисел с фиксированной точкой;
- подробная и доступная для понимания документация.

Из недостатков модуля можно выделить:

- отсутствие более сложных арифметических методов вроде деления и вычитания для чисел непозиционной системы счисления;
- арифметика непозиционной системы счисления используется внутри модуля, что делает ее недоступной для пользователя;
- нет возможности реализовать арифметику непозиционной системы счисления вне контекста конкретной схемы шифрования.

Модуль гомоморфного шифрования «HElib».

Модуль «HElib» предоставляет методы для шифрования данных по схемам BGV и CKKS, поддерживает выполнение сложной арифметики чисел НСС и

побитовые операции. Для выполнения математических действий над числами используется математический модуль «NTL».

Данный модуль обладает следующими преимуществами:

- высокий уровень оптимизации при работе с различными схемами шифрования;
- наличие методов сложной арифметики вроде векторизации и побитовых операций;
- поддержка низкоуровневых команд языка ассемблера для гомоморфного шифрования.

Из недостатков модуля можно выделить:

- сложная структура кода и настройка модуля;
- менее понятная документация;
- отсутствие более сложных арифметических методов вроде деления и вычитания для чисел непозиционной системы счисления;
- арифметика непозиционной системы счисления используется внутри модуля, что делает ее недоступной для пользователя;
- нет возможности реализовать арифметику непозиционной системы счисления вне контекста конкретной схемы шифрования.

Математический модуль «NTL».

Модуль «NTL» предоставляет структуры и алгоритмы для произведения высокоэффективных вычислений в области целочисленной и полиномиальной арифметики. Данный модуль не предоставляет методы для гомоморфного шифрования, однако имеет инструменты для их реализации.

Данный модуль обладает следующими преимуществами:

- оптимизация работы с большими числами и полиномами;
- поддержка модульной арифметики;
- является проверенным, используется во многих других криптографических модулях.

Из недостатков модуля можно выделить:

– не имеет явной реализации арифметики непозиционной системы счисления;

– не имеет интерфейса для работы с дробными числами;

– не предоставляет инструменты для шифрования.

1.1.3 Обоснование актуальности разработки программы

Разработанный в данной работе модуль непозиционной обработки цифровых данных будет включать в себя необходимые структуры, классы и функции сложной арифметики для обработки чисел системы остаточных классов для целочисленных и действительных чисел в формате фиксированной точки. Также, разработанный прикладной модуль будет иметь оптимизацию по количеству выделяемой памяти и количеству нормализаций без потери результата с помощью математических свойств модулей. Модуль будет оснащен удобным интерфейсом и сопровождается полнодоступной документацией с возможностью компиляции под различные платформы (Windows, Linux). На его основе можно будет реализовать более сложные платформы для гомоморфных вычислений, в том числе и облачных.

1.1.4 Постановка задачи

Требуется реализовать прикладной модуль, который будет соответствовать следующим функциональным требованиям:

– приведение целых чисел и чисел с фиксированной точкой к системе остаточных классов;

– дешифрация чисел системы остаточных классов на основе оснований системы;

– наличие сложных арифметических методов для выполнения гомоморфных вычислений над реализованными типами данных, в том числе сложения, умножения, вычитания и деления;

– возможность построить статические и динамические библиотеки под различные платформы;

– возможность задать пользовательские основания системы остаточных классов.

					ПГУ 1.090301.03.001 ПЗ	Лист
						12
Изм.	Лист	№ докум.	Подпись	Дата		

Разрабатываемый прикладной модуль должен соответствовать следующим техническим требованиям:

- удобный и интуитивно понятный интерфейс для взаимодействия с модулем;
- оптимизация вычислений на основе параллельной обработки независимых разрядов чисел;
- оптимизация затрачиваемой памяти на основе математических свойств модульной арифметики;
- возможность масштабирования модуля;
- разработанный модуль является библиотекой статической и динамической линковки;
- операционная система: Windows/Linux;
- размер скомпилированной библиотеки: 27 КБ ПЗУ.

Список экспортируемых элементов библиотеки:

- класс RNSuint, описывающий беззнаковый целочисленный тип данных в рамках системы остаточных классов;
- класс RNSint, описывающий знаковый целочисленный тип данных в рамках системы остаточных классов;
- класс RNSfxp, описывающий тип данных с фиксированной точкой в контексте системы остаточных классов;
- класс RNSCrypter, предоставляющий базовые инструменты для взаимного преобразования простых типов позиционной системы счисления и типов системы остаточных классов;
- класс RNSOverflowError, предоставляющий собой константное значение числа 256 в системе остаточных классов.

1.2 Выбор технологий и средств разработки приложения

В данном разделе будут описаны технологии и средства разработки, использованные при разработке программного модуля непозиционной обработки цифровых данных.

Ниже приведен список выбранных технологий и средств разработки:

- Assembly;
- C#;
- C/C++;
- Meson;
- Bazel;
- CMake;
- Bash;
- Kate;
- Visual Studio 2019;
- Visual Studio Code.

1.2.1 Выбор языка программирования

Язык программирования Assembly.

Assembly является низкоуровневым языком программирования, предоставляющим доступ к управлению всеми элементами вычислительной системы. Позволяет выполнять высокоэффективные вычисления и минимизировать размер исходного кода, однако не подходит для проектирования сложных решений, поскольку имеет высокую сложность разработки.

Язык ассемблера обладает следующими преимуществами:

- высокая производительность;
- прямой доступ к аппаратным ресурсам.

К недостаткам можно будет отнести:

- исходный код не переносится между разными архитектурами ВМ;
- сложность расширения проекта;
- низкий уровень абстракции.

Язык программирования C#.

C# – это современный высокоуровневый объектно-ориентированный язык программирования, разработанный компанией Microsoft. За счет объектно-ориентированного подхода позволяет выполнять проектирование громоздких проектов, однако является зависимым от платформы .NET Framework, если не брать во внимание .NET Core [4], а также имеет потенциальные проблемы с производительностью при сложной логике исходного кода.

Язык программирования C# обладает следующими преимуществами:

- высокий уровень абстракции;
- кроссплатформенность (.Net Core);
- парадигма объектно-ориентированного программирования;
- наличие автоматического сборщика мусора.

К недостатку можно отнести сравнительно низкую производительность по сравнению с более низкоуровневыми языками программирования, в том числе C++;

Язык программирования C++.

C++ – это высокоуровневый язык программирования, расширяющий базовый функционал языка C элементами объектно-ориентированного программирования [5] и другими инструментами. Данный язык программирования позволяет компилировать высокоэффективные программы и библиотеки благодаря возможности ручного управления памятью, а также поддерживает возможность многопоточного программирования.

Среди преимуществ языка программирования C++ можно выделить:

- высокая производительность;
- гибкость, сочетание низкоуровневого программирования и абстракций объектно-ориентированного программирования;
- парадигма объектно-ориентированного программирования;
- большое число поддерживаемых сторонних модулей.

Из недостатков можно выделить:

- ручное управление памятью;

- небезопасный программный код;
- сложность языка за счет обилия инструментов.

1.2.2 Выбор средств построения проекта

Meson – это система сборки проектов, зарекомендовавшая себя по скорости сборки, простоте использования и возможности сборки проектов под различные платформы.

Из преимуществ можно выделить:

- высокая скорость сборки проекта;
- встроенная система встраивания зависимостей.

Из недостатков можно выделить:

- сложные проекты требуют более сложной настройки;
- меньшая поддержка сообщества.

Bazel – это система сборки проектов с открытым исходным кодом, созданная компанией Google. Обеспечивает высокую скорость сборки проектов за счет кэширования и параллельной обработки модулей.

Из преимуществ можно выделить:

- высокая скорость сборки проекта;
- меньшая зависимость от настройки окружения.

Из недостатков можно выделить:

- сложность освоения;
- сложная конфигурация.

CMake – кроссплатформенный инструмент для построения C/C++ проектов, однако работает и с проектами на других языках программирования. Позволяет автоматизировать процесс линковки и сборки проекта. Является простым в освоении и идеально подойдет для построения решений под платформы Windows и Linux [6].

Из преимуществ можно выделить:

- гибкость настройки проекта;
- переносимые конфигурации сборки;
- поддержка во многих IDE;

– автоматизация сборки проектов.

Bash – скриптовый язык, являющийся специфичным для каждой платформы. Позволит автоматизировать запуск последовательных команд терминала, что еще больше упростит дальнейшую работу с CMake.

1.2.3 Выбор среды разработки

Kate – продвинутый текстовый редактор от сообщества KDE, имеющий открытый исходный код. Предоставляет такой функционал как подсветка синтаксиса языка, автодополнения на основе контекста, удобный интерфейс для менеджмента построенного решения и подключение LSP сервера для предварительной компиляции кода.

Visual Studio – среда разработки на платформе Windows от компании Microsoft, предоставляющая огромное количество инструментов, необходимых для написания, тестирования и отладки проектов. Имеет удобный интерфейс, подсветку кода, автодополнения на основе контекста и множество скачиваемых дополнений.

Visual Studio Code – это кроссплатформенная среда для разработки, разработанная компанией Microsoft.

1.3 Проектирование программного модуля

1.3.1 Архитектура программного модуля

На рисунке 1 представлена архитектура программного модуля непозиционной обработки цифровых данных в виде диаграммы классов.

Числа формата системы остаточных классов реализованы в программном модуле в виде классов RNSuint, RNSint и RNSfxp. Классы RNSint и RNSfxp наследуются от базового класса RNSuint, и включают в своем составе реализацию арифметики для каждого конкретного типа данных. В свою очередь, класс RNSuint имеет реализацию базовой арифметики разрядов чисел типа системы остаточных классов.



Рисунок 1 – Диаграмма классов

Помимо классов самих чисел СОК, также существует класс, хранящий пользовательские основания системы и константы. Этот же класс используется для взаимного преобразования чисел простейших типов с вышеприведенными RNSCrypter.

Исходя из вышесказанного можно выделить несколько основных этапов использования библиотеки:

- инициализация классов для хранения оснований, констант и выполнения преобразований;
- инициализация классов типов данных системы остаточных классов с помощью класса шифратора RNSCrypter;
- выполнение пользовательской арифметики в соответствии с ограничениями типов;

– нормализация и/или выполнение приведения к простейшему типу результата.

1.4 Разработка программного модуля

1.4.1 Математическое обоснование

Непозиционная система счисления на основе чисел системы остаточных классов строится вокруг китайской теоремы об остатках. В соответствии с которой, числа формата системы остаточных классов будут определены взаимно простыми модулями, называемыми базисами. Разряды таких чисел системы остаточных классов, будут определены однозначно для множества целых чисел без знака $[0; P)$, где P – мощность выбранной системы базисов, вычисляемая как произведение всех модулей системы.

К примеру, для взаимно простых оснований 3, 5 и 7 значение мощности системы остаточных классов P будет равно $3 * 5 * 7 = 105$, из чего следует, что такой набор оснований позволит закодировать числа в диапазоне $[0; 105)$.

1.4.1.1 Целочисленный тип без знака

Целочисленный тип без знака является базовой формой числа формата системы остаточных классов и не требует дополнительных манипуляций для реализации арифметики и обратного преобразования.

1.4.1.2 Приведение к типу формата СОК

Для приведения целого числа без знака к типу формата системы остаточных классов необходимо выполнить расчет каждого из разрядов числа, с помощью деления по модулю базиса, соответствующего разряда.

Для примера возьмём набор взаимно простых оснований 3, 5, 7 и целое число без знака 69_{10} . Для каждого из разрядов выполним деление по модулю базиса:

$$69 \% 3 = 0;$$

$$69 \% 5 = 4;$$

$$69 \% 7 = 6.$$

Таким образом, полная запись целого числа без знака в формате системы остаточных классов для базисов 3, 5, 7 будет: $\{0; 4; 6\}$.

1.4.1.3 Реализация арифметических операций

Для произведения операции сложения беззнаковых целых чисел формата системы остаточных классов необходимо произвести базовую арифметическую операцию сложения между соответствующими разрядами чисел СОК рассматриваемой суммы с учетом нормализации по базисам, где нормализация по базисам – это приведение разрядов числа формата системы остаточных классов к наименьшему делимому по модулю базиса.

К примеру, для взаимно простых базисов 3, 5, 7, а также исходных слагаемых 5_{10} и 10_{10} , представленных в системе остаточных классов как $\{2; 0; 5\}$ и $\{1; 0; 3\}$ сумма будет вычислена как:

$$(2 + 1) \% 3 = 0;$$

$$(0 + 0) \% 5 = 0;$$

$$(5 + 3) \% 7 = 1.$$

Полная запись результата будет соответствовать форме $\{0; 0; 1\}$. Результатом сложения слагаемых является число 15_{10} , что можно проверить с помощью приведения числа к формату системы остаточных классов:

$$15 \% 3 = 0;$$

$$15 \% 5 = 0;$$

$$15 \% 7 = 1.$$

Результат приведения типов совпал с результатом сложения исходных слагаемых, следовательно, алгоритм работает верно.

Для произведения операции умножения беззнаковых целых чисел формата системы остаточных классов необходимо произвести базовую арифметическую операцию умножения, аналогично описанной ранее операции сложения с учетом нормализации по базисам.

Для взаимно простых базисов 3, 5, 7, а также чисел 6_{10} и 11_{10} , представленных в системе остаточных классов как $\{0; 1; 6\}$ и $\{2; 1; 4\}$ произведение будет вычислено как:

$$(0 * 2) \% 3 = 0;$$

$$(1 * 1) \% 5 = 1;$$

$$(6 * 4) \% 7 = 3;$$

Полная запись результата будет соответствовать форме $\{0; 1; 3\}$.
Результатом произведения множителей является число 66_{10} , что можно проверить с помощью приведения числа к формату системы остаточных классов:

$$66 \% 3 = 0;$$

$$66 \% 5 = 1;$$

$$66 \% 7 = 3;$$

Результат приведения типов совпал с результатом произведения исходных множителей, следовательно, алгоритм работает верно.

Операция вычитания производится почти аналогично операции сложения за небольшим изменением. Для произведения операции вычитания между двумя числами системы остаточных классов, необходимо произвести базовую арифметическую операцию вычитания между соответствующими разрядами этих чисел с прибавлением значения модуля и последующей нормализацией результата по базисам.

К примеру, для набора базисов 3, 5, 7, а также чисел 50_{10} и 12_{10} , представленных в системе остаточных классов как $\{2; 0; 1\}$ и $\{0; 2; 5\}$ разность будет вычислена как:

$$(2 - 0 + 3) \% 3 = 2;$$

$$(0 - 2 + 5) \% 5 = 3;$$

$$(1 - 5 + 7) \% 7 = 3.$$

Полная запись результата будет соответствовать форме $\{2; 3; 3\}$.
Результатом разности данных чисел является число 38_{10} , что можно проверить с помощью приведения числа к формату системы остаточных классов:

$$38 \% 3 = 2;$$

$$38 \% 5 = 3;$$

$$38 \% 7 = 3.$$

Результат приведения типов совпал с результатом разности исходных чисел, следовательно, алгоритм работает верно.

Операция деления в рамках чисел системы остаточных классов без потери значащего результата возможна только если деление производится нацело, другими словами без остатка. В следствие чего, операция деления в рамках данной дипломной работы будет реализована без фактического произведения операции деления чисел системы остаточных классов. Алгоритм деления описан далее в составе пункта типа фиксированной точки.

1.4.1.4 Приведение к простейшему типу

Для приведения числа формата системы остаточных классов к простейшему типу, в данном случае к целочисленному типу без знака, необходимо предварительно выполнить вычисление базовых чисел системы остаточных классов в рамках пользовательских оснований и мощности системы. Базовыми числами системы будут считаться такие числа, что имеют один разряд, равный единице, в то время как все остальные разряды равны нулю.

Например, для системы с базисами 3, 5 и 7 и мощностью системы 105 такими числами будут:

$$\{1; 0; 0\} = 70_{10};$$

$$\{0; 1; 0\} = 21_{10};$$

$$\{0; 0; 1\} = 15_{10}.$$

Конечным шагом преобразования будет сложение произведений базовых чисел системы со значением разрядов и последующим делением по модулю мощности выбранной системы.

Для примера возьмем те же основания с мощностью системы равной 105, а также число 69_{10} , представленное в системе остаточных классов с учетом выбранных базисов как $\{0; 4; 6\}$. Тогда, в соответствии с описанным выше алгоритмом, число из формата системы остаточных классов будет приведено к целому типу без знака как:

$$0 * 70 + 4 * 21 + 6 * 15 = 174;$$

$$174 \% 105 = 69.$$

Результат приведения числа из системы остаточных классов является верным, поскольку полученное целое число соответствует таковому до приведения к типу системы остаточных классов, в следствие чего можно сделать вывод о том, что преобразование работает верно.

1.4.1.5 Целочисленный тип со знаком

Существует несколько способов реализации знаковых чисел в пределах системы остаточных классов [7]. В настоящей дипломной работе будет рассмотрен способ хранения знакового числа с помощью введения дополнительного базиса равного двум в дополнение к пользовательским базисам.

Базис равный двум в составе набора простейших базисов соответственно увеличивает мощность системы оснований P в два раза. Идея состоит в том, чтобы искусственно сместить систему счисления до середины диапазона соответствующего набора базисов, другими словами, принять за начало отсчета середину диапазона. Числа системы остаточных классов, содержащие в себе смещение диапазона, будут называться искусственной формой.

Таким образом, после приведения числа искусственной формы к целочисленному типу ранее описанным алгоритмом, необходимо будет вычесть из результата целое число, соответствующее половине диапазона выбранной системы базисов.

Для примера возьмём систему с базисами 2, 3, 5, 7, также положительное и отрицательное числа -5_{10} и 13_{10} . Тогда данные числа будут приведены к искусственной форме как:

$$\begin{aligned} 5 &= \{1; 2; 0; 5\}; \\ \{1; 0; 0; 0\} - \{1; 2; 0; 5\} &= \{0; 1; 0; 2\}; \\ 13 &= \{1; 1; 3; 6\}; \\ \{1; 0; 0; 0\} + \{1; 1; 3; 6\} &= \{0; 1; 3; 6\}. \end{aligned}$$

В данном случае число $\{1; 0; 0; 0\}$ являет собой середину диапазона выбранных базисов.

Для проверки правильности результата приведения к искусственной форме выполним приведение к целочисленному типу со знаком для первого числа:

$$0 * 105 + 1 * 70 + 0 * 126 + 2 * 120 = 310;$$

$$310 \% 210 = 100;$$

$$100 - 105 = -5.$$

Для второго числа:

$$0 * 105 + 1 * 70 + 3 * 126 + 6 * 120 = 1168;$$

$$1168 \% 210 = 118;$$

$$118 - 105 = 13.$$

Результаты приведения обоих чисел из искусственной формы к знаковому целочисленному типу совпали с ожидаемыми, следовательно, данный способ действительно подходит для организации работы с числами формата системы остаточных классов.

В настоящей дипломной работе базис двойки будет отброшен. В таком случае все арифметические операции будут идентичны таковым в системе остаточных классов для беззнакового целочисленного типа. Изменению подвергся только алгоритм приведения знакового целочисленного типа в систему остаточных классов к простейшему, что делает эти два типа взаимно преобразуемыми с накладыванием некоторого ограничения в виде возможности получения отрицательного результата для беззнакового типа данных, либо обратной ситуации. Оба варианта потери значимых данных будут являться частным случаем переполнения или, другими словами, перехода в смежную часть диапазона со сменой порядка четности. При этом, так как добавления базиса двойки как такового не происходит, мощность системы с учетом текущих базисов остаётся прежней.

В таком случае, идея введения знакового типа в систему остаточных классов сводится не к удвоению ее исходного диапазона, а к разделению исходного диапазона на поддиапазоны положительных и отрицательных чисел. При этом, исходная точка системы отсчета не смещается, в следствие чего такой способ не обязует вводить в систему искусственную форму числа.

Для приведения к знаковому целочисленному типу в рамках системы остаточных классов будет достаточно либо выполнить вычитание между нулем и

исходным числом в системе остаточных классов методом вычитания беззнакового целочисленного типа, либо выполнить аналогичное таковому у беззнакового целочисленного типа приведение к типу системы остаточных классов. Выбор алгоритма будет зависеть от знака приводимого числа.

Рассмотрим приведение к простейшему типу для данного алгоритма. Для базисов системы 3, 5, 7, а также отрицательного числа

-16 алгоритм приведения к знаковому целочисленному типу будет:

$$16 = \{1; 1; 2\};$$

$$\{0; 0; 0\} - \{1; 1; 2\} = \{2; 4; 5\};$$

$$2 * 70 + 4 * 21 + 5 * 15 = 299.$$

Добавляем к общей сумме основных чисел системы половину диапазона, сдвигая число на значение середины диапазона, а затем, после нормализации по мощности системы, вычтем значение середины диапазона системы:

$$299 + 105 / 2 = 351;$$

$$351 \% 105 = 36;$$

$$36 - 105 / 2 = -16.$$

Результат приведения числа в системе остаточных классов к знаковому целочисленному типу совпал с заданным исходным значением, следовательно, данный способ подходит для введения знакового целочисленного типа в рамках системы остаточных классов.

1.4.1.6 Тип фиксированной точки

Для реализации типа данных фиксированной точки в рамках системы остаточных классов критически важно наличие операции деления, в том числе для случаев, когда деление нацело невозможно. Из-за специфики системы остаточных классов, для некоторых чисел системы результатом деления будет не одно, а несколько чисел, что не подходит для реализуемого программного модуля.

Для решения этой проблемы в общую систему вводится работа с дробями, где и числитель, и знаменатель – это числа системы остаточных классов, относящиеся к одному члену арифметической операции. Это решение позволяет заменить операции деления операциями умножения между числителями и

знаменателями, однако накладывает ряд дополнительных ограничений для уже устоявшихся арифметических методов.

В соответствии с базовыми математическими законами логики нельзя сложить или выполнить вычитание между двумя дробями с различающимися знаменателями. Таким образом, помимо увеличения количества вычислений, возрастает и риск получения переполнения в рамках диапазона системы, поскольку для чисел системы остаточных классов нет методов сравнения чтобы упрощать дроби.

Такая система вводится, начиная с беззнакового целочисленного типа данных, поскольку тот так же не имеет алгоритма деления, из-за чего все три типа данных становятся идентичными по способу хранения разрядов дроби.

Как и в случае со знаковым типом данных, изменению подверглись алгоритмы приведения типов. Таким образом, для преобразования исходного числа в тип с фиксированной точкой в рамках системы остаточных классов стало необходимо сместить запятую числа на определенное количество символов по базису с последующим выполнением алгоритма приведения для знакового типа для числителя и беззнакового для знаменателя, а при приведении к простому типу с плавающей запятой выполнить уже выведенные алгоритмы приведения для знаковых чисел, но с сохранением остатка от деления.

Стоит отметить, что система остаточных классов не имеет такого свойства как потеря остатка при делении, из-за чего использование целочисленных типов вне зависимости от знака может быть неочевидным.

1.4.1.7 Нормализация по модулю 256

В рамках гомоморфной арифметики хранение базисов системы остаточных классов на сторонней вычислительной машине нивелирует все положительные стороны непозиционной системы счисления, из-за чего возникает необходимость выполнения всех ранее описанных арифметических операций без знания базисов. Исключением в данном случае будут приведения типов, поскольку они будут производиться на доверенных вычислительных машинах.

Для решения этой проблемы существует накопительное свойство разрядов чисел системы остаточных классов, в следствие модульной эквивалентности чисел. Например, число 6 эквивалентно числу 33 по модулю 3. В соответствии с этим свойством выведем новые алгоритмы арифметических операций.

Проще всего будет модифицировать операцию сложения, поскольку действительно необходимо будет убрать из общего алгоритма только операцию деления по модулю на базисы.

Например, возьмём числа 2_{10} , 5_{10} и 8_{10} в формате системы остаточных классов с базисами 3, 5 и 7. В виде чисел системы остаточных классов данные числа будут представлены как $\{2; 2; 2\}$, $\{2; 0; 5\}$ и $\{2; 3; 1\}$. Тогда сложение без предварительной нормализации по базисам будет произведено как:

$$2 + 2 + 2 = 6;$$

$$2 + 0 + 3 = 5;$$

$$2 + 5 + 1 = 8;$$

В последствии, при нормализации результата мы получаем $\{0; 0; 1\}$, что переводится в позиционную систему как 15_{10} . Сумма исходных чисел в позиционной системе счисления с основанием 10 совпадает с полученным результатом, следовательно, алгоритм может использоваться для сложения чисел системы остаточных классов.

Стоит отметить, что вычислительные машины имеют физические ограничения по количеству памяти, из-за чего выполнять ряд арифметических операций без нормализаций, а если точнее, без знания базисов, становится затруднительным. Когда при выполнении арифметической операции на каком-либо из разрядов числа системы остаточных классов возникает переполнение, порядок смены четности ломается, и данные безвозвратно теряются.

Для решения этой проблемы был разработан алгоритм, выполняющий частичную нормализацию разрядов чисел системы остаточных классов с помощью приведенного к системе остаточных классов числа со значением 256. Идея состоит в том, чтобы при возникновении переполнения восстанавливать порядок смены четности на разрядах. Для этого необходимо взять разрядность

разрядов числа системы остаточных классов в два раза больше значимой части, чтобы даже при перемножении предельно больших для разряда чисел значения не терялись.

Для проведения исследования рассмотрим предельные значения самого малого типа данных в большинстве современных вычислительных машин – 1 байт, кодирующий значения в диапазоне $[0; 255]$. Допустим, мы имеем базисы системы 3, 5 и 7, а также предельно допустимое на разрядах число $\{255; 255; 255\}$. Если выполнить нормализацию по базисам и привести к позиционной системе счисления, то результатом будет число 45_{10} , что в системе остаточных классов будет выглядеть как $\{0; 0; 3\}$. В соответствии с имеющейся базовой арифметикой разрядов числа мы предполагаем, что следующее число будет иметь значение $\{1; 1; 4\}$, однако по переполнению мы получаем значение $\{0; 0; 0\}$, что соответствует значению нуля в позиционной системе счисления.

Рассмотрим потерянные значения разрядов. Следующим за числом $\{0; 0; 3\}$ значением должно идти число $\{1; 1; 4\}$, однако из-за переполнения результату не хватает значения $\{1; 1; 4\}$ на разрядах, из чего можно сделать вывод, что при единичном переполнении от 255 к 0, происходит потеря значимого результата в размере $\{1; 1; 4\}$, что при приведении к позиционной системе счисления будет значить 46_{10} . Стоит отметить, что число 46 является эквивалентным к числу 256 по модулю мощности набора текущих базисов системы остаточных классов. Таким образом, при переполнении одного из разрядов числа, оно теряет значение, эквивалентное таковому на разрядах числа 256 в формате системы остаточных классов.

При наличии нескольких переполнений подряд, что часто случается при выполнении операции умножения, происходит потеря указанного ранее числа количеством, равным целой части от деления на 256, следовательно, общий алгоритм нормализации по модулю для каждого разряда будет выглядеть как:

$$m_i = hm_i * ov_i + lm_i,$$

где m_i – значение разряда числа системы остаточных классов, hm_i – старшая часть исходного значения разряда числа, ov_i – разряд числа 256 в системе остаточных классов, а lm_i – младшая часть исходного значения разряда числа.

Стоит также отметить, что в некоторых случаях результатом коррекции может стать число имеющее переполнение по модулю 256. В таких случаях необходимо выполнить нормализацию по модулю 256 несколько раз.

С учетом вышесказанного можно также сказать, что алгоритм умножения будет так же легко модифицировать, поскольку он не имеет каких-либо дополнительных ограничений кроме ограничения по памяти для разрядов.

Например, возьмём те же числа 2_{10} , 5_{10} и 8_{10} в формате системы остаточных классов с базисами 3, 5 и 7. В виде чисел системы остаточных классов данные числа будут представлены как $\{2; 2; 2\}$, $\{2; 0; 5\}$ и $\{2; 3; 1\}$. Тогда умножение без предварительной нормализации по базисам будет произведено как:

$$2 * 2 * 2 = 8;$$

$$2 * 0 * 3 = 0;$$

$$2 * 5 * 1 = 10.$$

При нормализации результата мы получаем $\{2; 0; 3\}$, что переводится в позиционную систему как 80_{10} . Произведение исходных чисел в позиционной системе счисления с основанием 10 совпадает с полученным результатом, следовательно, алгоритм может использоваться для умножения чисел системы остаточных классов.

Алгоритм деления, как уже было сказано ранее, заменен сочетанием работы с дробями и алгоритмом умножения.

Алгоритм вычитания в текущем списке гомоморфных арифметических функций является одним из сложно реализуемых, поскольку в исходном алгоритме дополнительно присутствует значение базиса для защиты от переполнения, однако, небольшая модификация исходного алгоритма нормализации по модулю 256 решает данную проблему.

При разности разрядов с переполнением происходит очередная потеря значения, применяется алгоритм нормализации по модулю 256, однако только

единожды, и отсекается старшая часть разряда. Такой подход позволяет сохранить значение при вычитании даже без знания базисов системы.

Для примера возьмем числа 15_{10} и 1_{10} с базисами 3, 5 и 7. В формате системы остаточных классов эти числа будут представлены как $\{0; 0; 1\}$ и $\{1; 1; 1\}$. Тогда разность будет выполнена как:

$$\{0; 0; 1\} - \{1; 1; 1\} = \{32767; 32767; 0\}.$$

Выполняем единичную нормализацию по модулю 256 для каждого из разрядов:

$$(255 * 1 + 255) \% 256 = 254;$$

$$(255 * 1 + 255) \% 256 = 254;$$

$$(0 * 4 + 0) \% 256 = 0.$$

При нормализации результата мы получаем $\{2; 4; 0\}$, что переводится в позиционную систему как 14_{10} . Разность исходных чисел в позиционной системе счисления с основанием 10 совпадает с полученным результатом, следовательно, алгоритм может использоваться для вычитания чисел системы остаточных классов.

Таким образом, описанной выше математической теории достаточно для реализации программного модуля непозиционной обработки цифровых данных.

1.4.2 Описание программного модуля

Файл `.gitignore` – это конфигурационный файл системы контроля версий Git, необходимый для указания на то, какие из файлов не должны отслеживаться в текущем репозитории.

Файл `CMakeLists.txt` – это конфигурационный файл системы сборки проектов CMake, необходимый для определения стратегии построения проекта.

Файл `RNSCrypter.hpp` – это заголовочный файл, содержащий в себе объявление класса `RNSCrypter` с перечислением его методов.

Файл `RNSCrypter.cpp` – это файл исходного кода, содержащий в себе реализацию методов класса `RNSCrypter`.

Класс `RNSCrypter` существует для нескольких целей:

- генерация констант для вычисления: простейших чисел системы пользовательских базисов, мощности системы;
- приведение простых типов к типам формата системы остаточных классов;
- приведение типов формата системы остаточных классов к простым типам;
- выполнение нормализации разрядов чисел системы остаточных классов.

Метод `calcConsts` используется для подсчета всех констант, необходимых для приведения типов.

Перегруженные методы `encode` используются для приведения простых типов к формату чисел системы остаточных классов.

Перегруженные методы `decode` используются для приведения чисел формата системы остаточных классов к простым типам данных.

Метод `normalize` используется для произведения нормализации чисел системы остаточных классов в соответствии с сохраненными базисами.

Файл `RNSuint.hpp` – это заголовочный файл, содержащий в себе объявление класса `RNSuint` с перечислением его методов и частичной реализацией.

Файл `RNSuint.cpp` – это файл исходного кода, содержащий в себе реализацию методов класса `RNSuint`.

Класс `RNSuint` являет собой беззнаковые целые числа в формате системы остаточных классов. Этот класс является основой для арифметики зависимых классов.

Методы `add`, `sub`, `mul`, `div` используются для произведения гомоморфных арифметических действий над числами системы остаточных классов.

Метод `overflowCorrection` используется для произведения нормализации чисел системы остаточных классов по модулю 256 без использования базисов системы.

Метод `singleOverflowCorrection` – модифицированный метод `overflowCorrection`, выполняющий всего одну итерацию нормализации с последующим отбрасыванием старшей значимой части разрядов.

Файл `RNSint.hpp` – это заголовочный файл, содержащий в себе объявление класса `RNSint` с перечислением его методов и реализацией.

Файл RNSfxr.hpp – это это заголовочный файл, содержащий в себе объявление класса RNSfxr с перечислением его методов и реализацией.

1.4.4 Тестирование и отладка

В процессе работы с программным модулем могут возникнуть некоторые исключительные ситуации, влияющие на конечный результат преобразований. Например, пользователь библиотеки может не произвести нормализацию по переполнению до того, как часть результата будет утеряна, либо не выполнить базовую нормализацию чисел системы остаточных классов, из-за чего результат так же будет искажен. Для таких ситуаций в классах предусмотрен вывод отладочной информации и выбрасывание исключений.

					ПГУ 1.090301.03.001 ПЗ	Лист
						32
Изм.	Лист	№ докум.	Подпись	Дата		

2 Охрана труда

Современное законодательство в области охраны труда при работе с электронно-вычислительными машинами (ЭВМ) выдвигает следующие требования [8]:

Помещение, в котором осуществляется работа с ПК, должно соответствовать следующим требованиям:

- освещение должно соответствовать стандартам, а в случае отсутствия естественного освещения должны быть предоставлены расчеты, доказывающие безопасность работы без него;
- оконные проемы должны иметь регулируемые устройства, такие как жалюзи или занавеси;
- компьютеры не должны размещаться в местах, где есть риск попадания влаги.
- рабочие места с компьютерами должны быть оборудованы защитным заземлением;
- в помещениях с компьютерами следует проводить ежедневную влажную уборку и проветривание каждый час;
- рекомендуемая температура воздуха в помещении с компьютерами составляет $+19^{\circ}\text{C}$ - $+21^{\circ}\text{C}$;
- рабочий стол должен быть спроектирован таким образом, чтобы обеспечить оптимальное расположение используемого оборудования в соответствии с выполняемой работой, разрешается использование различных типов рабочих столов, соответствующих современным стандартам эргономики;
- в помещениях с компьютерами следует проводить ежедневную влажную уборку и проветривание каждый час.

Рабочие места с персональными компьютерами должны быть организованы следующим образом:

- при организации рабочих мест с компьютерами необходимо учитывать, что расстояние между рабочими столами, на которых установлены

видеомониторы, должно быть не менее 2,0 метров в направлении между ними и не менее 1,2 метра между боковыми поверхностями видеомониторов;

– размещение компьютеров должно предусматривать расстояние не менее 1 метра от источников тепла. Следует избегать прямого попадания солнечного света на них;

– запрещается размещать рабочие места с компьютерами рядом с силовыми кабелями, высоковольтными трансформаторами и другим оборудованием, способным создавать помехи в работе компьютеров;

– при установке компьютеров следует избегать прохождения интерфейсных и электропитания кабелей по полу. В случае необходимости, их следует закрыть защитными коробками;

– экран видеомонитора должен находиться от глаз пользователя на расстоянии 600-700 мм, но не ближе 500 мм с учетом размеров алфавитно-цифровых знаков и символов;

– системный блок и монитор следует разместить так, чтобы расстояние от вентиляционных отверстий этих устройств до стены (или стенки мебели) составляло не менее 10 сантиметров;

– искусственное освещение в помещениях для эксплуатации компьютеров должно осуществляться системой общего равномерного освещения. Освещенность на поверхности рабочего стола в зоне размещения ПК должна быть 300-500 люкс. Освещение не должно создавать бликов на поверхности экрана. Освещенность поверхности экрана не должна быть более 300 люкс.

В случае работы на компьютере в течение 8-часовой смены следует предусматривать перерывы в соответствии с категорией работ:

- для работников I категории перерывы устанавливаются через 2 часа после начала смены и через 2 часа после обеденного перерыва продолжительностью 15 минут каждый;

- работники II категории должны иметь перерыв через 2 часа после начала смены и через 1,5 - 2,0 часа после обеденного перерыва продолжительностью 15 минут каждый или продолжительностью 10 минут через каждый час работы;

					ПГУ 1.090301.03.001 ПЗ	Лист
						34
Изм.	Лист	№ докум.	Подпись	Дата		

- для работников III категории рекомендуется устанавливать перерыв через 1,5 - 2,0 часа после начала смены и также через 1,5 - 2,0 часа после обеденного перерыва продолжительностью 20 минут каждый или продолжительностью 15 минут через каждый час работы.

При 12-часовой смене регламентированные перерывы должны быть предусмотрены в первые 8 часов работы так же, как и при 8-часовой смене. В течение последних 4 часов работы, независимо от категории и вида работ, необходимо предусматривать перерыв каждый час продолжительностью 15 минут (таблица 1).

Таблица 1 – Время регламентированных перерывов в зависимости от категории работы

Категория работы	Уровень нагрузки за рабочую смену при видах работ с компьютером			Суммарное время регламентированных перерывов, мин.	
	группа А, количество знаков	группа Б, количество знаков	группа В, час.	при 8-часовой смене	при 12-часовой смене
I	До 20000	До 15000	До 2,0	30	70
II	До 40000	До 30000	До 4,0	50	90
III	До 60000	До 40000	До 6,0	70	120

Заключение

В ходе выполнения данной выпускной квалификационной работы был спроектирован и разработан программный модуль непозиционной обработки цифровых данных.

В результате был проведен глубокий анализ принципов организации современной информационной безопасности. Перед началом разработки модуля был проведен анализ предметной области, проанализированы косвенные аналоги уже разработанных средств гомоморфного шифрования, выбраны необходимые средства и инструменты для разработки собственного программного модуля.

В результате было принято решение разработать программный модуль на языке программирования C/C++. Для построения проекта программного модуля с учетом требования к кроссплатформенности был выбран CMake. В качестве среды разработки выбраны Visual Studio 2019 и Visual Code.

На основе приведенных выше технологий, удалось спроектировать, разработать и протестировать программный модуль, со следующим функционалом:

- задание пользовательских базисов системы остаточных классов;
- использование арифметических методов над числами формата системы остаточных классов без раскрытия базисов;
- работа с беззнаковым целочисленным типом данных в рамках системы остаточных классов;
- работа со знаковым целочисленным типом данных в рамках системы остаточных классов;
- работа с числами с фиксированной точкой в рамках системы остаточных классов;
- приведение сложных типов за счет полиморфизма.

Разработанный в данной выпускной квалификационной работе модуль на текущий момент содержит весь необходимый базовый функционал, предъявляемый требованиями к выбранному типу шифрования. На данном этапе

существования проекта он, как и множество аналогичных сторонних библиотек, нуждается в доработке и расширении функционала. Созданный программный модуль, призван расширить возможности существующих вычислительных систем в области шифрования и работы с числами больших размерностей.

					ПГУ 1.090301.03.001 ПЗ	Лист
						37
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Документация к библиотеке гомоморфного шифрования Microsoft SEAL. [Электронный ресурс] / Режим доступа: <https://www.microsoft.com/en-us/research/project/microsoft-seal/>, свободный (дата обращения 02.05.25);
2. Документация к библиотеке гомоморфного шифрования HElib. [Электронный ресурс] / Режим доступа: <https://homenc.github.io/HElib/>, свободный (дата обращения 02.05.25);
3. Документация к библиотеке NTL. [Электронный ресурс] / Режим доступа: <https://libntl.org/>, свободный (дата обращения 02.05.25);
4. М.Е. Фленов. Библия С#. 5-е издание. – М.: Издательство «БХВ», 2022. – 464 с.
5. Мартин Ридди. Разработка API для C++. 2-е издание. – М.: Морган Кауфман, 2024. – 648 с.
6. Дэвид Кохин, Кристиан Штурм. Руководство к Линуксу для разработчика ПО. 1-е издание. – М.: Издательство «Пакт», 2024. – 300 с.
7. И.Я. Акушерский, Д.И. Юдицкий. Машинная арифметика в остаточных классах. 1-е издание. – М.: Издательство «Советское радио», 1968. 439 с.
8. Инструкция по охране труда при работе с ЭВМ. [Электронный ресурс] / Режим доступа: https://kpfu.ru/portal/docs/F_1086113514/1815.pdf/, свободный (дата обращения 10.05.25).

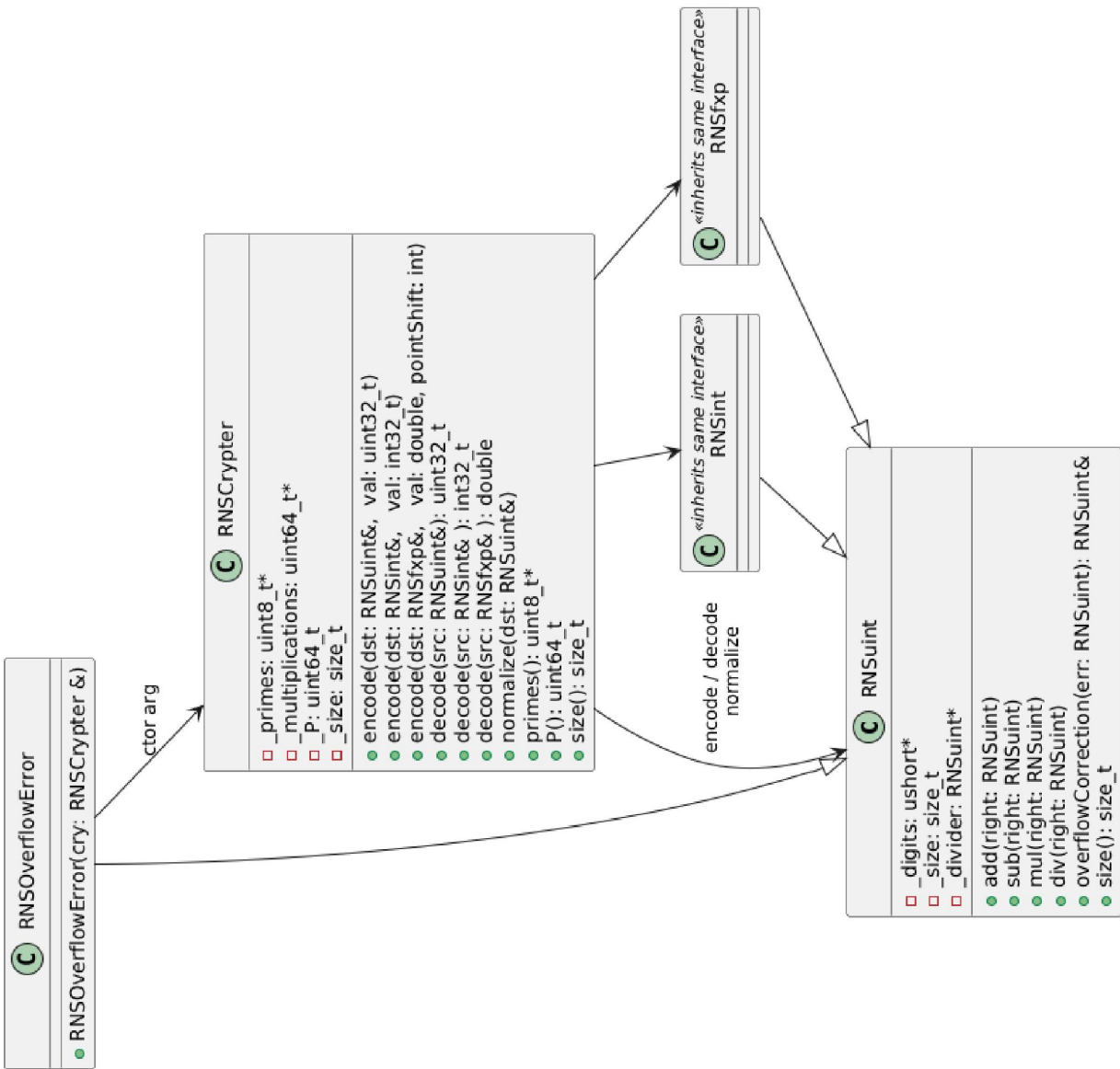
Приложение А

(обязательное)

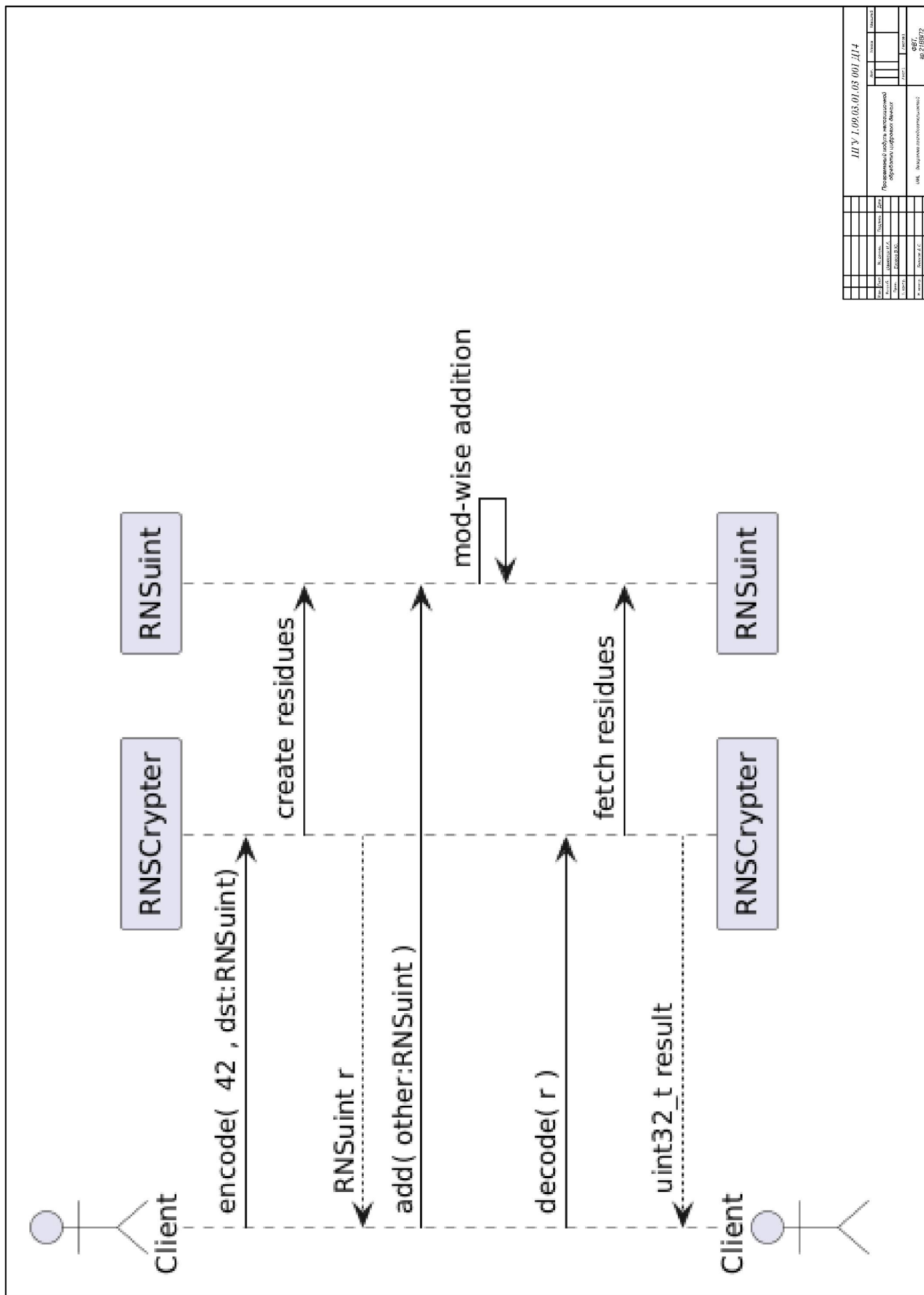
Программный модуль непозиционной обработки цифровых данных
UML-диаграммы

					ПГУ 1.090301.03.001 ПЗ	Лист
						39
Изм.	Лист	№ докум.	Подпись	Дата		

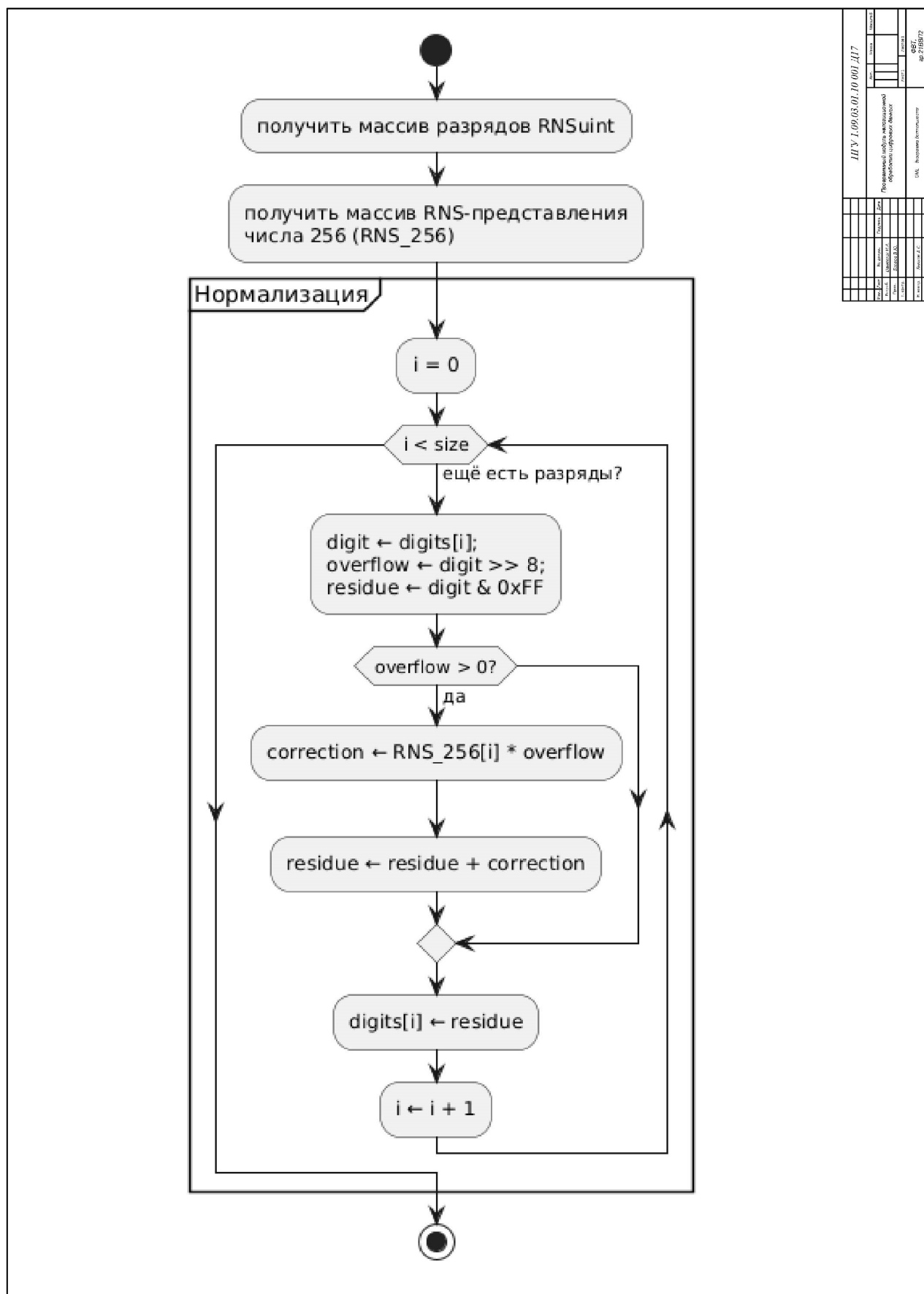
Приложение А.1 – UML – диаграмма классов

[illegible]

Приложение А.2 – UML – диаграмма последовательностей

[illegible]

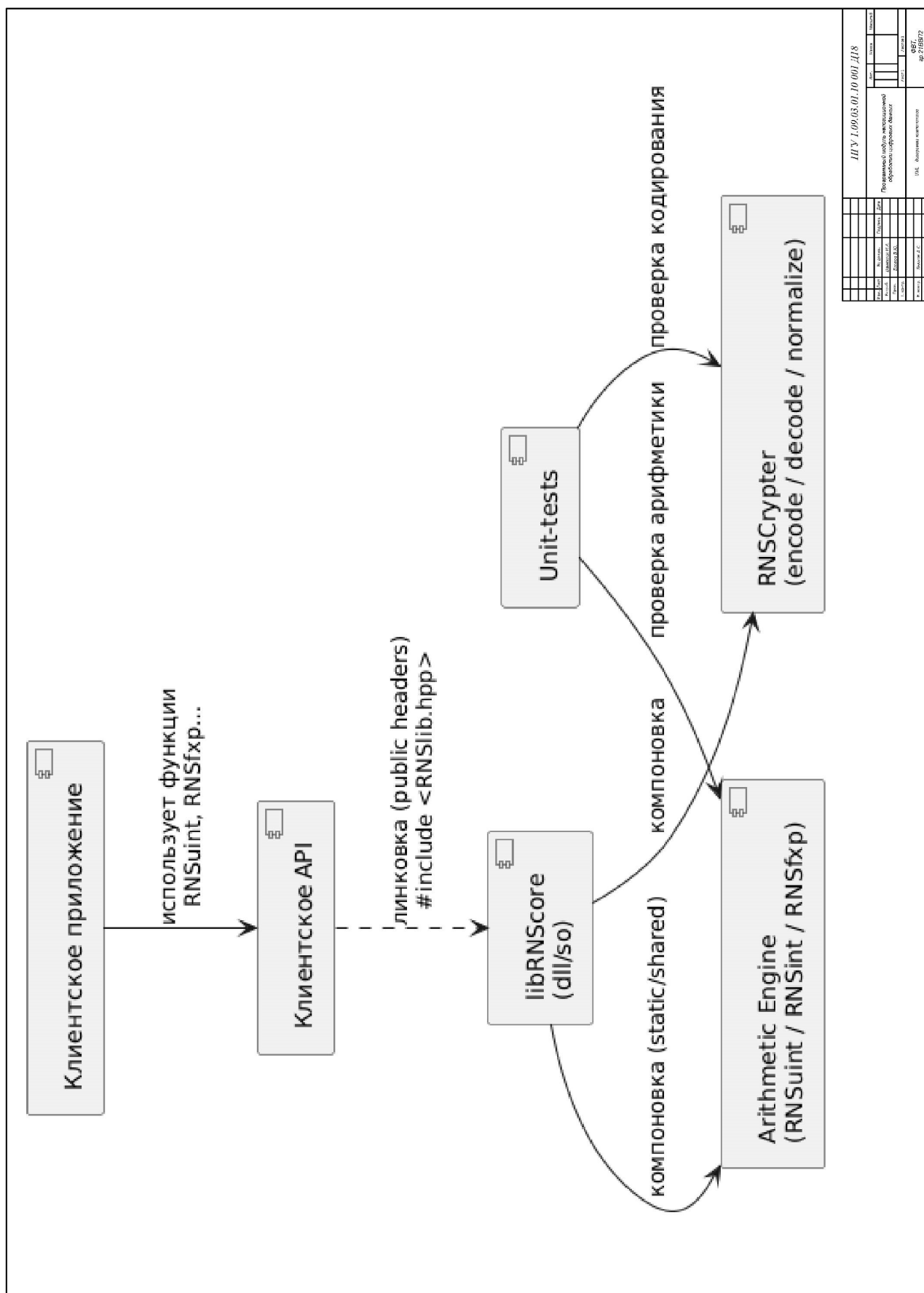
Приложение А.3 – UML – диаграмма деятельности



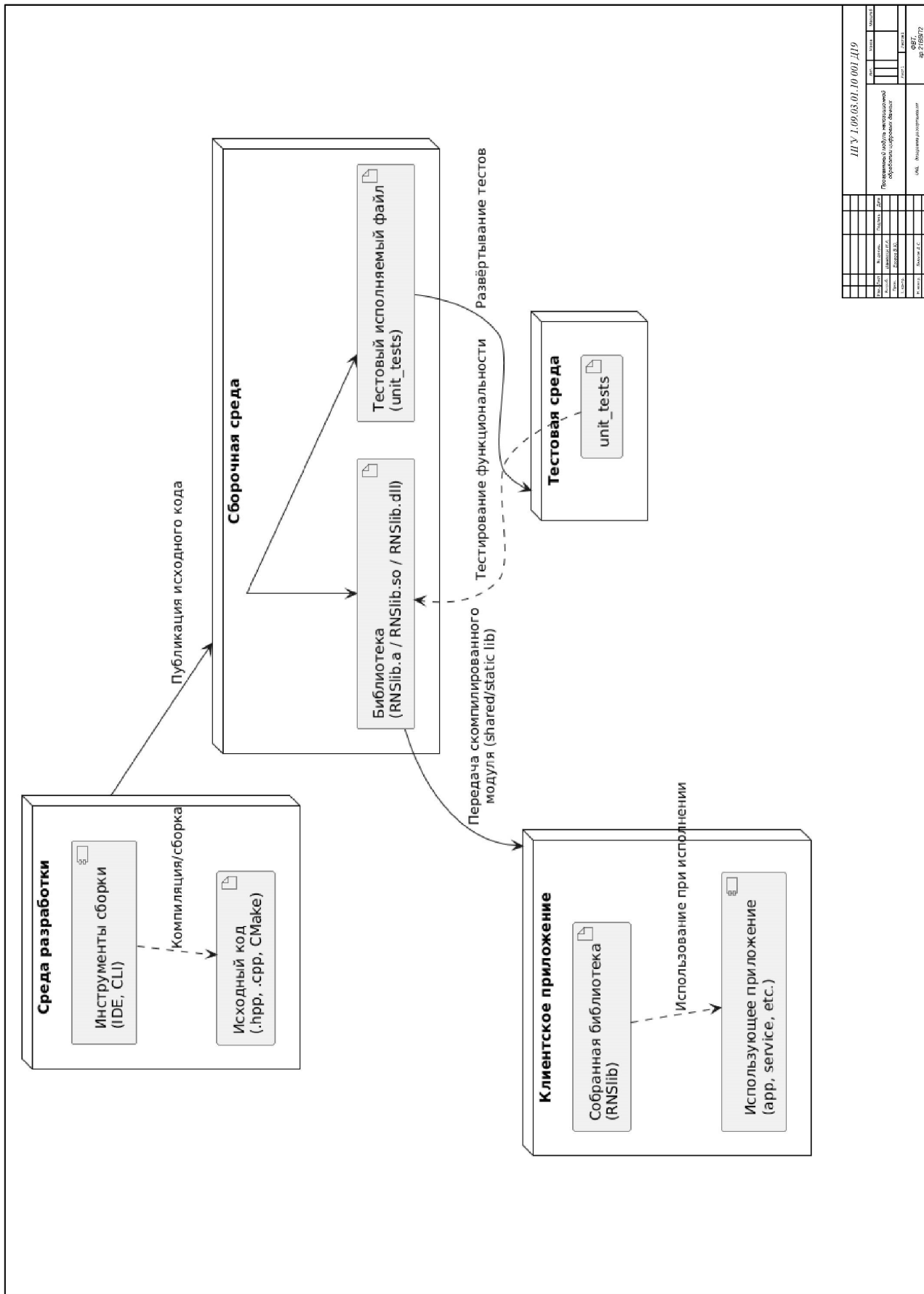
ИЗУ 1.090301.03.001 ПЗ		ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ
ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ
ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ
ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ
ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ
ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ
ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ
ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ
ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ
ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ	ИЗУ 1.090301.03.001 ПЗ

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.4 – UML – диаграмма компонентов

[illegible]

Приложение А.5 – UML – диаграмма развёртывания



Приложение Б

(Обязательное)

Программный модуль непозиционной обработки цифровых данных

Листинги программы

					ПГУ 1.090301.03.001 ПЗ	Лист
						45
Изм.	Лист	№ докум.	Подпись	Дата		

Файл RNSCrypter.hpp:

```
#pragma once
#include <cstdint>

class RNSuint;
class RNSint;
class RNSfxp;

// The class is used to store calculation constants and
// encrypt/decrypt RNS numbers.
class RNSCrypter
{
private:
    uint8_t* _primes;

    uint64_t* _multiplications;
    uint64_t _P;
    size_t _size;

private:
    // Function calculates constants
    void calcConsts();

    int32_t uniDecode(const RNSuint* rnsNum, bool sign) const;

public:
    // Default constructors.
    RNSCrypter() = delete;

    // Constructor sets constants by "primes" array.
    RNSCrypter(const uint8_t* primes, size_t _size);

    // Copy constructor
    RNSCrypter(const RNSCrypter& src);

    RNSCrypter(RNSCrypter&& src) noexcept;

public:
    // Returns a pointer to an array of bases.
    const uint8_t* primes() const { return _primes; };

    uint64_t P() const { return _P; }

    // Returns a count of bases.
    size_t size() const { return _size; };

public:
    // Encodes uint32 number to RNS code.
    // Writes result in the "dst" array.
```

					ПГУ 1.090301.03.001 ПЗ	Лист
						46
Изм.	Лист	№ докум.	Подпись	Дата		

```

void encode(RNSuint& dst, uint32_t decNum) const;

// Encodes int32 number to RNS code.
// Writes result in the "dst" array.
void encode(RNSint& dst, int32_t decNum) const;

// Encodes long float number to RNS code.
// Writes result in the "dst" array.
void encode(RNSfxp& dst, double lfNum, int pointShift) const;

// Decodes RNS code to uint32 number.
uint32_t decode(const RNSuint& rnsNum) const;

// Decodes RNS code to int32 number.
int32_t decode(const RNSint& rnsNum) const;

// Decodes RNS code to long float number.
double decode(const RNSfxp& rnsNum) const;

void normalize(RNSuint& dst) const;

private:
// Polinomial reverse module
static uint32_t reverseModule(uint32_t a, uint32_t m)
{
    uint32_t result{ 1 };
    for (int i = 0; i < m - 2; i++) result *= a;
    return result % m;
}

public:
void operator=(const RNSCrypter& src);

void operator=(RNSCrypter&& src) noexcept;

public:
// Deconstructor
~RNSCrypter()
{
    if (_primes != nullptr) delete[] _primes;
    if (_multiplications != nullptr) delete[] _multiplications;
}
};

```

Файл RNSCrypter.cpp:

```

#include "RNSLib/RNSCrypter.hpp"
#include "RNSLib/RNSuint.hpp"
#include "RNSLib/RNSint.hpp"
#include "RNSLib/RNSfxp.hpp"

```

```

#include <memory>

// Constructors

RNSCrypter::RNSCrypter(const uint8_t* primes, size_t size) :
    _size(size)
{
    _primes = new uint8_t[_size];
    _multiplications = new uint64_t[_size];

    memcpy(_primes, primes, _size * sizeof(uint8_t));

    calcConsts();
}

RNSCrypter::RNSCrypter(const RNSCrypter& src) : _size(src._size),
    _P(src._P)
{
    _primes = new uint8_t[_size];
    _multiplications = new uint64_t[_size];

    memcpy(_primes, src._primes, _size * sizeof(uint8_t));
    memcpy(_multiplications, src._multiplications, _size *
sizeof(uint32_t));
}

RNSCrypter::RNSCrypter(RNSCrypter&& src) noexcept : _size(src._size),
    _P(src._P)
{
    _primes = nullptr;
    _multiplications = nullptr;

    std::swap(_primes, src._primes);
    std::swap(_multiplications, src._multiplications);
}

void RNSCrypter::encode(RNSuint& dst, uint32_t decNum) const
{
    for (size_t i = 0; i < _size; i++)
        dst[i] = decNum % _primes[i];
}

void RNSCrypter::encode(RNSint& dst, int32_t decNum) const
{
    if (decNum >= 0)
    {
        for (size_t i = 0; i < _size; i++)
            dst[i] = decNum % _primes[i];
    }
    else
    {
        RNSuint error(_size);
    }
}

```

```

        RNSint tmpRNS(_size);
        encode(tmpRNS, -decNum);
        encode(error, 256);

        for (size_t i = 0; i < _size; i++)
            dst[i] = 0;

        dst.sub(tmpRNS);
        dst.singleOverflowCorrection(error);
        normalize(dst);
    }
}

void RNSCrypter::encode(RNSfxp& dst, double lfNum, int pointShift)
const
{
    RNSuint divider(_size);

    uint32_t base = 10;
    base = std::pow(base, pointShift);
    lfNum *= base;

    encode((RNSint&)dst, (int32_t)lfNum);
    encode(divider, base);
    dst.div(divider);
    normalize(dst);
}

// Constants calculation

void RNSCrypter::calcConsts() {
    _P = _primes[0];

    for (size_t i = 1; i < _size; i++)
    {
        _P *= _primes[i];
    }

    for (size_t i = 0; i < _size; i++)
    {
        _multiplications[i] = _P / _primes[i];

        for (auto j = 1; j < _primes[i]; j++) {
            if (_multiplications[i] * j % _primes[i] == 1)
            {
                _multiplications[i] *= j;
                break;
            }
        }
    }
}

int32_t RNSCrypter::uniDecode(const RNSuint* rnsNum, bool sign) const

```

```

{
    int64_t preresult = 0;
    int32_t divider = (rnsNum->hasDivider() ? uniDecode(rnsNum->divider(), sign) : 1);

    for (size_t i = 0; i < _size; i++)
    {
        preresult += rnsNum->operator[](i) * _multiplications[i];
    }

    if (sign)
        preresult += (_P >> 1);

    return (int32_t)(preresult % _P - (_P >> 1) * sign) / divider;
}

// Decoders

uint32_t RNSCrypter::decode(const RNSuint& rnsNum) const
{
    uint64_t preresult = 0;
    uint32_t divider = (rnsNum.hasDivider() ?
decode(*rnsNum.divider()) : 1);

    for (size_t i = 0; i < _size; i++)
    {
        preresult += rnsNum[i] * _multiplications[i];
    }

    return (uint32_t)(preresult % _P) / divider;
}

int32_t RNSCrypter::decode(const RNSint& rnsNum) const
{
    return uniDecode(&rnsNum, true);
}

double RNSCrypter::decode(const RNSfxp& rnsNum) const
{
    int64_t preresult = 0;
    int64_t predivider = 0;
    int32_t divider = 1;

    for (size_t i = 0; i < _size; i++)
    {
        preresult += rnsNum[i] * _multiplications[i];
        predivider += rnsNum.divider()->operator[](i) *
_multiplications[i];
    }

    preresult += (_P >> 1);
    predivider += (_P >> 1);

```

```

        divider = (predivider % _P - (_P >> 1));

        return (int32_t)(preresult % _P - (_P >> 1)) / (double)divider;
    }

void RNSCrypter::normalize(RNSuint& dst) const
{
    for (size_t i = 0; i < this->_size; i++)
    {
        dst[i] %= _primes[i];
    }

    if (dst.hasDivider())
    {
        for (size_t i = 0; i < this->_size; i++)
        {
            dst.divider()->operator[](i) %= _primes[i];
        }
    }
}

// Operators

void RNSCrypter::operator=(const RNSCrypter& src)
{
    if (this == &src) return;

    if (_size != src._size)
    {
        delete[] _primes;
        delete[] _multiplications;

        _primes = new uint8_t[src._size];
        _multiplications = new uint64_t[src._size];
    }

    _size = src._size;
    _P = src._P;

    memcpy(_primes, src._primes, _size * sizeof(uint8_t));
    memcpy(_multiplications, src._multiplications, _size *
sizeof(uint32_t));
}

void RNSCrypter::operator=(RNSCrypter&& src) noexcept
{
    _size = src._size;
    _P = src._P;

    std::swap(_primes, src._primes);
    std::swap(_multiplications, src._multiplications);
}

```

```
}
```

Файл RNSuint.hpp:

```
#pragma once  
#include <fstream>
```

```
class RNSuint  
{  
private:  
    unsigned short int* _digits;  
    size_t _size;  
  
protected:  
    RNSuint* _divider;  
  
public:  
    RNSuint(size_t size = 0);  
  
    RNSuint(const unsigned char* digits, size_t size);  
  
    virtual ~RNSuint();  
  
    RNSuint(const RNSuint& src);  
  
    RNSuint(RNSuint&& src) noexcept;  
  
    void operator=(const RNSuint& src);  
  
    void operator=(RNSuint&& src) noexcept;  
  
public:  
    size_t size() const { return _size; };  
    bool hasDivider() const { return _divider != nullptr; }  
  
    const unsigned short int* digits() const { return _digits; }  
    unsigned short int* digits() { return _digits; }  
  
    const RNSuint* divider() const { return _divider; }  
    RNSuint* divider() { return _divider; }  
  
public:  
    void setDigits(const unsigned char* digits, size_t size);  
  
public:  
    // Local adding operation for RNS numbers.  
    virtual void add(const RNSuint& rightRnsNum);  
  
    // Local subliming operation for RNS numbers.  
    virtual void sub(const RNSuint& rightRnsNum);
```

					ПГУ 1.090301.03.001 ПЗ	Лист
						52
Изм.	Лист	№ докум.	Подпись	Дата		

```

        // Local multiplying operation for RNS numbers.
        virtual void mul(const RNSuint& rightRnsNum);

        // Local multiplying operation for RNS numbers.
        virtual void div(const RNSuint& rightRnsNum);

public:
    // Overflow correction
    RNSuint& overflowCorrection(const RNSuint& overflowErrorValue);

    RNSuint& singleOverflowCorrection(const RNSuint&
overflowErrorValue);

    bool overflowFlag(size_t index) { return (_digits[index] >> 8) !=
0; }

public:
    void input(std::ifstream& fin);

    void output(std::ofstream& fout) const;

    std::string toString() const;

    friend std::string to_string(const RNSuint& item) { return
item.toString(); };

public:
    inline friend RNSuint& operator+(const RNSuint& left, const
RNSuint& right)
    {
        RNSuint* result = new RNSuint(left);
        result->add(right);

        return *result;
    }

    inline friend RNSuint& operator-(const RNSuint& left, const
RNSuint& right)
    {
        RNSuint* result = new RNSuint(left);
        result->sub(right);

        return *result;
    }

    inline friend RNSuint& operator*(const RNSuint& left, const
RNSuint& right)
    {
        RNSuint* result = new RNSuint(left);
        result->mul(right);

        return *result;
    }

```

```

    }

    inline friend RNSuint& operator/(const RNSuint& left, const
RNSuint& right)
    {
        RNSuint* result = new RNSuint(left);
        result->div(right);

        return *result;
    }

    inline void operator+=(const RNSuint& right) { this->add(right);
}

    inline void operator-=(const RNSuint& right) { this->sub(right);
}

    inline void operator*=(const RNSuint& right) { this->mul(right);
}

    inline void operator/=(const RNSuint& right) { this->div(right);
}

    inline RNSuint& operator++()
    {
        for (size_t i = 0; i < _size; i++)
        {
            if (_divider == nullptr)
                ++_digits[i];
            else
                _digits[i] += (*_divider)[i];
        }
        return *this;
    }

    inline RNSuint& operator++(int)
    {
        RNSuint& result = *new RNSuint{ *this };
        this->operator++();

        return result;
    }

    inline RNSuint& operator--()
    {
        for (size_t i = 0; i < _size; i++)
        {
            if (_divider == nullptr)
                --_digits[i];
            else
                _digits[i] -= (*_divider)[i];
        }
        return *this;
    }

```

```

    }

    inline RNSuint& operator--(int)
    {
        RNSuint& result = *new RNSuint{ *this };
        this->operator--();

        return result;
    }

    bool operator==(const RNSuint& right) const
    {
        if (_size != right._size) return false;
        if ((_divider != nullptr) ^ right.hasDivider()) return false;

        for (auto i = 0; i < _size; i++)
        {
            if (_digits[i] != right._digits[i] || (_divider !=
nullptr && (*_divider)[i] != (*right._divider)[i])) return false;
        }

        return true;
    };

    bool operator!=(const RNSuint& right) const { return !(*this ==
right); };

    uint16_t operator[](size_t index) const
    {
        if (index >= _size) throw
std::exception("RNSVector.operator[]: index out of range.");
        return _digits[index];
    }

    uint16_t& operator[](size_t index)
    {
        if (index >= _size) throw
std::exception("RNSVector.operator[]: index out of range.");
        return _digits[index];
    }

public:
    friend std::ostream& operator<<(std::ostream& out, const RNSuint&
item)
    {
        for (size_t i = 0; i < item.size(); i++) out << item[i];
        return out;
    }

    friend std::istream& operator>>(std::istream& in, RNSuint& item)
    {
        for (size_t i = 0; i < item.size(); i++) in >> item[i];
        return in;
    }

```

```

    }
};

```

Файл RNSuint.cpp:

```

#include "RNSLib/RNSuint.hpp"
#include <vector>
#include <string>

RNSuint::RNSuint(size_t size) : _size(size), _digits(nullptr),
    _divider(nullptr)
{
    if (_size > 0)
    {
        _digits = new uint16_t[_size];
        memset(_digits, 0, sizeof(uint16_t) * _size);
    }
}

RNSuint::RNSuint(const unsigned char* digits, size_t size) : _size(0),
    _digits(nullptr), _divider(nullptr)
{
    setDigits(digits, size);
}

RNSuint::~RNSuint()
{
    if (_digits != nullptr) delete[] _digits;
    if (_divider != nullptr) delete _divider;
}

RNSuint::RNSuint(const RNSuint& src) : _size(src._size),
    _digits(nullptr), _divider(nullptr)
{
    if (_size > 0)
    {
        _digits = new uint16_t[_size];
        memcpy(_digits, src._digits, sizeof(uint16_t) * _size);

        if (src._divider != nullptr)
            _divider = new RNSuint(*src._divider);
    }
}

RNSuint::RNSuint(RNSuint&& src) noexcept : _size(src._size),
    _digits(nullptr), _divider(nullptr)
{
    std::swap(_digits, src._digits);
    std::swap(_divider, src._divider);
}

```

```

void RNSuint::operator=(const RNSuint& src)
{
    if (this == &src || (_size == 0 && src._size == 0)) return;

    if (src._size == 0)
    {
        delete[] _digits;
        if (_divider != nullptr) delete _divider;

        _size = 0;
        _digits = nullptr;
        _divider = nullptr;

        return;
    }

    if (_size != src._size)
    {
        if (_digits != nullptr) delete[] _digits;
        if (_divider != nullptr) delete _divider;

        _size = src._size;
        _digits = new uint16_t[_size];

        _divider = (src.hasDivider() ? new RNSuint(_size) : nullptr);
    }

    memcpy(_digits, src._digits, sizeof(uint16_t) * _size);
    if (src.hasDivider() && _divider == nullptr) _divider = new
RNSuint(_size);
    _divider->operator=(*src._divider);
}

void RNSuint::operator=(RNSuint&& src) noexcept
{
    _size = src._size;
    std::swap(_digits, src._digits);
    std::swap(_divider, src._divider);
}

void RNSuint::setDigits(const unsigned char* digits, size_t size)
{
    if (_size == 0 && size == 0) return;

    if (_size != size)
    {
        if (_size > 0)
        {
            delete[] _digits;
            _digits = nullptr;
        }

        _size = size;
    }
}

```

```

        if (size > 0)
        {
            _digits = new uint16_t[size];
        }
    }

    if (size > 0)
    {
        memcpy(_digits, digits, sizeof(uint16_t) * size);
    }
}

// Arithmetical operations

void RNSuint::add(const RNSuint& rightRnsNum)
{
    if (_size != rightRnsNum._size) throw
std::exception("RNSuint.add: Different vector sizes.");

    if (_divider != nullptr || rightRnsNum._divider != nullptr)
    {
        const auto* rightDivider = rightRnsNum._divider;
        if (_divider == nullptr)
        {
            _divider = new RNSuint(*rightDivider);
            for (size_t i = 0; i < _size; i++) _digits[i] *=
(*_divider)[i];
        }
        else if (_divider != rightDivider)
        {
            RNSuint tmpRight{ *_divider };
            tmpRight.mul(rightRnsNum);

            if (rightDivider != nullptr)
            {
                _divider->mul(*rightDivider);
                this->mul(*rightDivider);
            }

            for (auto i = 0; i < _size; i++)
                _digits[i] += tmpRight._digits[i];

            return;
        }
    }

    for (auto i = 0; i < _size; i++)
        _digits[i] += rightRnsNum._digits[i];
}

void RNSuint::sub(const RNSuint& rightRnsNum)

```

```

{
    if (_size != rightRnsNum._size) throw
std::exception("RNSuint.sub: Different vector sizes.");

    if (_divider != nullptr || rightRnsNum._divider != nullptr)
    {
        const auto* rightDivider = rightRnsNum._divider;
        if (_divider == nullptr)
        {
            _divider = new RNSuint(*rightDivider);
            this->mul(*_divider);
        }
        else if (_divider != rightDivider)
        {
            RNSuint tmpRight{ *_divider };
            tmpRight.mul(rightRnsNum);

            if (rightDivider != nullptr)
            {
                _divider->mul(*rightDivider);
                this->mul(*rightDivider);
            }

            for (auto i = 0; i < _size; i++)
                _digits[i] -= tmpRight._digits[i];

            return;
        }
    }

    for (auto i = 0; i < _size; i++)
        _digits[i] -= rightRnsNum._digits[i];
}

void RNSuint::mul(const RNSuint& rightRnsNum)
{
    if (_size != rightRnsNum._size) throw
std::exception("RNSuint.mul: Different vector sizes.");

    if (_divider != nullptr || rightRnsNum._divider != nullptr)
    {
        const auto* rightDivider = rightRnsNum._divider;

        if (_divider == nullptr)
            _divider = new RNSuint(*rightDivider);
        else if (rightDivider != nullptr)
            _divider->mul(*rightDivider);
    }

    for (auto i = 0; i < _size; i++)
        _digits[i] *= rightRnsNum._digits[i];
}

```

```

void RNSuint::div(const RNSuint& rightRnsNum)
{
    if (_size != rightRnsNum._size) throw
std::exception("RNSuint.div: Different vector sizes.");

    if (!_divider)
    {
        _divider = new RNSuint(_size);

        for (size_t i = 0; i < _size; i++)
            (*_divider)[i] = 1;
    }

    for (size_t i = 0; i < _size; i++)
    {
        (*_divider)[i] *= rightRnsNum._digits[i];
        if (rightRnsNum.hasDivider()) _digits[i] *=
(*rightRnsNum._divider)[i];
    }
}

RNSuint& RNSuint::overflowCorrection(const RNSuint&
overflowErrorValue)
{
    for (size_t i = 0; i < _size; i++)
    {
        _digits[i] = (_digits[i] & 0x00ff) + (_digits[i] >> 8) *
overflowErrorValue._digits[i];
        if ((_digits[i] & 0xff00) > 0) --i;
    }

    if (_divider != nullptr)
    {
        auto& divider = *_divider;

        for (size_t i = 0; i < _size; i++)
        {
            divider[i] = (divider[i] & 0x00ff) + (divider[i] >> 8) *
overflowErrorValue._digits[i];
            if ((divider[i] & 0xff00) > 0) --i;
        }
    }

    return *this;
}

RNSuint& RNSuint::singleOverflowCorrection(const RNSuint&
overflowErrorValue)
{
    for (auto i = 0; i < _size; i++)
    {
        _digits[i] = (_digits[i] & 0x00ff) + (_digits[i] >> 8) *
overflowErrorValue._digits[i];
    }
}

```

```

        _digits[i] = _digits[i] & 0x00ff;
    }

    if (_divider != nullptr)
    {
        auto& divider = *_divider;

        for (size_t i = 0; i < _size; i++)
        {
            divider[i] = (divider[i] & 0x00ff) + (divider[i] >> 8) *
overflowErrorValue._digits[i];
            if ((divider[i] & 0xff00) > 0) --i;
        }
    }

    return *this;
}

void RNSuint::input(std::ifstream& fin)
{
    for (auto i = 0; i < _size; i++) fin >> _digits[i];
}

void RNSuint::output(std::ofstream& fout) const
{
    for (auto i = 0; i < _size; i++)
    {
        fout << _digits[i];

        if (i < _size - 1) fout << " ";
    }
}

std::string RNSuint::toString() const
{
    std::string result{ "" };

    for (size_t i = 0; i < _size; i++)
    {
        result += std::to_string(_digits[i]);
        if (i != _size - 1) result.push_back(' ');
    }

    return result;
}

```

Файл RNSint.hpp:

```

#pragma once
#include <RNSLib/RNSuint.hpp>

```

					ПГУ 1.090301.03.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

```

class RNSint : public RNSuint
{
public:
    RNSint(size_t size = 0) : RNSuint(size) {}

    RNSint(const RNSint& src) : RNSuint(src) {}

    RNSint(RNSint&& src) noexcept : RNSuint(std::move(src)) {}

    void operator=(const RNSint& src)
    {
        RNSuint::operator=(src);
    }
    void operator=(RNSint&& src) noexcept
    {
        RNSuint::operator=(std::move(src));
    }

    inline friend RNSint& operator+(const RNSint& a, const RNSint& b)
    {
        RNSint& result = *new RNSint(a);
        result.add(b);
        return result;
    }

    inline friend RNSint& operator-(const RNSint& a, const RNSint& b)
    {
        RNSint& result = *new RNSint(a);
        result.sub(b);
        return result;
    }

    inline friend RNSint& operator*(const RNSint& a, const RNSint& b)
    {
        RNSint& result = *new RNSint(a);
        result.mul(b);
        return result;
    }

    inline friend RNSint& operator/(const RNSint& a, const RNSint& b)
    {
        RNSint& result = *new RNSint(a);
        result.div(b);
        return result;
    }

    inline void operator+=(const RNSint& right) { this->add(right); }
    inline void operator-=(const RNSint& right) { this->sub(right); }
    inline void operator*=(const RNSint& right) { this->mul(right); }
    inline void operator/=(const RNSint& right) { this->div(right); }

    inline RNSint& operator++()

```

```

{
    RNSuint::operator++();
    return *this;
}

inline RNSint& operator++(int)
{
    RNSint& result = *new RNSint{ *this };
    this->operator++();
    return result;
}

inline RNSint& operator--()
{
    RNSuint::operator--();
    return *this;
}

inline RNSint& operator--(int)
{
    RNSint& result = *new RNSint{ *this };
    this->operator--();
    return result;
}

bool operator==(const RNSint& right) const
{
    return RNSuint::operator==(right);
};

bool operator!=(const RNSint& right) const { return !(*this ==
right); };

public:
    // Overflow correction
    RNSint& overflowCorrection(const RNSuint& overflowErrorValue)
    {
        RNSuint::overflowCorrection(overflowErrorValue);
        return *this;
    }

    RNSint& singleOverflowCorrection(const RNSuint&
overflowErrorValue)
    {
        RNSuint::singleOverflowCorrection(overflowErrorValue);
        return *this;
    }
};

```

Файл RNSfxp.hpp:

					ПГУ 1.090301.03.001 ПЗ	Лист
						63
Изм.	Лист	№ докум.	Подпись	Дата		

```

#pragma once
#include <RNSLib/RNSuint.hpp>

class RNSfxp : public RNSuint
{
public:
    RNSfxp(size_t size = 0) : RNSuint(size) {}

    RNSfxp(const RNSfxp& src) : RNSuint(src) {}

    RNSfxp(RNSfxp&& src) noexcept : RNSuint(std::move(src)) {}

    void operator=(const RNSfxp& src)
    {
        RNSuint::operator=(src);
    }
    void operator=(RNSfxp&& src) noexcept
    {
        RNSuint::operator=(std::move(src));
    }

    inline friend RNSfxp& operator+(const RNSfxp& a, const RNSfxp& b)
    {
        RNSfxp& result = *new RNSfxp(a);
        result.add(b);
        return result;
    }

    inline friend RNSfxp& operator-(const RNSfxp& a, const RNSfxp& b)
    {
        RNSfxp& result = *new RNSfxp(a);
        result.sub(b);
        return result;
    }

    inline friend RNSfxp& operator*(const RNSfxp& a, const RNSfxp& b)
    {
        RNSfxp& result = *new RNSfxp(a);
        result.mul(b);
        return result;
    }

    inline friend RNSfxp& operator/(const RNSfxp& a, const RNSfxp& b)
    {
        RNSfxp& result = *new RNSfxp(a);
        result.div(b);
        return result;
    }

    inline void operator+=(const RNSfxp& right) { this->add(right); }
    inline void operator-=(const RNSfxp& right) { this->sub(right); }
    inline void operator*=(const RNSfxp& right) { this->mul(right); }

```

					ПГУ 1.090301.03.001 ПЗ	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

```

inline void operator/=(const RNSfxp& right) { this->div(right); }

inline RNSfxp& operator++()
{
    RNSuint::operator++();
    return *this;
}

inline RNSfxp& operator++(int)
{
    RNSfxp& result = *new RNSfxp{ *this };
    this->operator++();
    return result;
}

inline RNSfxp& operator--()
{
    RNSuint::operator--();
    return *this;
}

inline RNSfxp& operator--(int)
{
    RNSfxp& result = *new RNSfxp{ *this };
    this->operator--();
    return result;
}

bool operator==(const RNSfxp& right) const
{
    return RNSuint::operator==(right);
};

bool operator!=(const RNSfxp& right) const { return !(*this ==
right); };

public:
    // Overflow correction
    RNSfxp& overflowCorrection(const RNSuint& overflowErrorValue)
    {
        RNSuint::overflowCorrection(overflowErrorValue);
        return *this;
    }

    RNSfxp& singleOverflowCorrection(const RNSuint&
overflowErrorValue)
    {
        RNSuint::singleOverflowCorrection(overflowErrorValue);
        return *this;
    }
};

```

Файл RNSOverflowError.hpp:

```
#pragma once
#include "RNSuint.hpp"
#include "RNSCrypter.hpp"

class RNSOverflowError : public RNSuint
{
public:
    RNSOverflowError() = delete;

    RNSOverflowError(const RNSCrypter& crypter) :
    RNSuint(crypter.size())
    {
        crypter.encode(*this, 256);
    };
};
```

					ПГУ 1.090301.03.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(Обязательное)

Программный модуль непозиционной обработки цифровых данных

Презентация

					ПГУ 1.090301.03.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

ПРОГРАММНЫЙ МОДУЛЬ НЕПОЗИЦИОННОЙ ОБРАБОТКИ ЦИФРОВЫХ ДАННЫХ

Автор работы: студент группы 21ВВП2

Ванюшин И.А.

Научный руководитель: к.т.н., доцент кафедры ВТ

Егоров В.Ю.

Цель работы

- Разработка программного обеспечения, предоставляющего функции для выполнения гомоморфных вычислений в непозиционной системе счисления
- ◇ Сокращение оснований непозиционной системы
- ◇ Оптимизация последовательных арифметических операций
- ◇ Сужение значений разрядов чисел СОК
- ◇ Представление различных типов данных в НСС

Актуальность

- Новые подходы к эффективной обработке чисел большой разрядности на ЭВМ
- Основа для построения более комплексной систем гомоморфных вычислений

Косвенные аналоги

microsoft/SEAL

Microsoft SEAL is an easy-to-use and powerful homomorphic encryption library.



44

Contributors

112

Issues

4k

Stars

736

Forks



homenc/HElib

HElib is an open-source software library that implements homomorphic encryption. It supports the BGV scheme with bootstrapping and the Approximate...



30

Contributors

176

Issues

3k

Stars

762

Forks



libntt/ntt



2

Contributors

14

Issues

290

Stars

56

Forks



Стек технологий



C/C++



CMake



Visual Studio

В процессе разработки были решены следующие задачи:

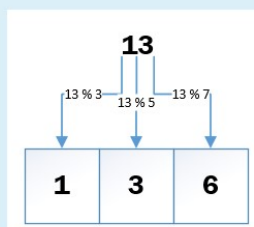
- Представление целых чисел без знака в непозиционной системе счисления
- Представление целых знаковых чисел в непозиционной системе счисления
- Представление чисел с фиксированной точкой в непозиционной системе счисления

Основные классы модуля

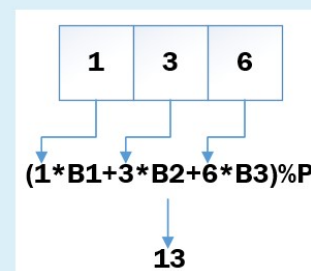
- **RNSCrypter** – выполняет хранение базисов, расчёт констант для декодирования и предоставляет методы приведения типов
- **RNSuint** – представление целых чисел без знака в СОК
- **RNSint** – представление целых знаковых чисел в СОК
- **RNSfxp** – представление чисел с фиксированной точкой в СОК
- **RNSOverflowError** – константа для нормализации разрядов чисел СОК по модулю 256

Целочисленный тип без знака

Диапазон значимых чисел системы $[0; P-1]$, где P - мощность набора оснований НСС

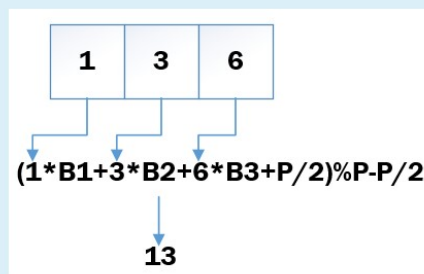


1	3	6
1	1	1



Целочисленный тип со знаком

Диапазон значимых чисел системы $[-(P-1)/2; (P-1)/2]$, где P - мощность набора оснований НСС

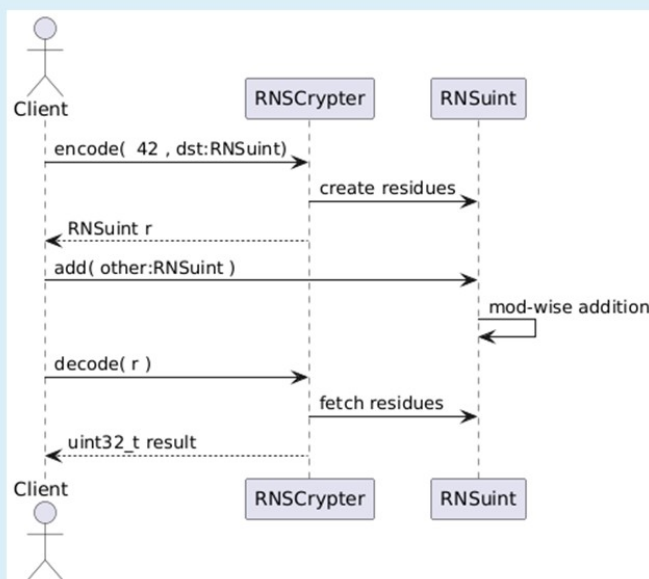


Тип с фиксированной точкой

Диапазон значимых чисел системы $[-(P-1)/2; (P-1)/2]$, где P - мощность набора оснований НСС

1	3	6
1	1	1

Диаграмма последовательности



Заключение и перспективы развития

Реализация:

- Программный модуль, поддерживающий представления трех типов данных, а также операций сложения, вычитания, деления и умножения в системе остаточных классов.

Дальнейшее развитие:

- Раширение функционала ПО: операции сравнения, расширения оснований, добавление тригонометрических функций.
- Уход от текущей реализации приведения чисел СОК к простым типам и реализации операции деления.