Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Удмуртский государственный университет»
Институт права, социального управления и безопасности
Кафедра информационной безопасности в управлении

М. М. Гайсин

МЕТОДИКА ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ»

Учебно-методическое пособие

Ижевск
2026

УДК 004.42(075.8)
ББК 32.973р30
Г147

Рекомендовано к изданию Учебно-методическим советом УдГУ

Рецензент: канд. техн. наук, ген. директор ООО НПП «Ижинформпроект» **В. Н. Цыркин**

Гайсин М. М.

Г147 Методика выполнения лабораторных работ по дисциплине «Технологии и методы программирования» : учеб.-метод. пособие / М. М. Гайсин . – Ижевск : Удмуртский университет, 2026. – 1,6 Мб. – Текст : электронный.

Учебно-методическое пособие подготовлено в соответствии с федеральным государственным образовательным стандартом высшего образования по направлению подготовки 10.03.01 «Информационная безопасность» и 10.05.05 «Безопасность информационных технологий в правоохранительной сфере». В нем приводятся теоретические основы и методические рекомендации по выполнению лабораторных работ обучающихся, требования к оформлению отчета работ и образцы их заполнения.

Пособие предназначено для обучающихся очной и очно-заочной форм обучения направления подготовки 10.03.01 «Информационная безопасность» и 10.05.05 «Безопасность информационных технологий в правоохранительной сфере» Института права, социального управления и безопасности, а также для преподавательского состава

Минимальные системные требования:
Celeron 1600 Mhz; 128 Мб RAM; Windows XP/7/8 и выше, 8x DVD-ROM
разрешение экрана 1024×768 или выше; программа для просмотра pdf.

Гайсин Марс Марсельевич
Методика выполнения лабораторных работ по дисциплине «Технологии и методы программирования»
Учебно-методическое пособие

Подписано к использованию 17.09.2026
Объем электронного издания 1,6 Мб
Издательский центр «Удмуртский университет»
426034, г. Ижевск, ул. Ломоносова, д. 4Б, каб. 021
Тел.: +7(3412)263-751 E-mail: editorial@udsu.ru

Содержание

Введение

Программный продукт является результатом некоего технологического процесса, в реализации которого участвует коллектив программистов. Технология программирования изучает технологические процессы и порядок их прохождения – стадии (с использованием знаний, методов и средств). Знания, методы и средства могут использоваться в разных процессах и, следовательно, технологиях. Чтобы разобраться в существующих технологиях программирования и определить основные тенденции их развития, целесообразно рассматривать эти технологии в историческом контексте, выделяя основные этапы развития программирования как науки. **Первый этап** – «стихийное» программирование. Он охватывает период от момента появления первых вычислительных машин до середины 60-х годов XX в. В этот период практически отсутствовали сформулированные технологии и программирование фактически было искусством. Первые программы имели простейшую структуру: собственно программа и обрабатываемые ею данные. Программы писали в машинных кодах. С появлением ассемблеров стало возможно использование вместо двоичных или 16-ричных кодов символических имён данных и мнемоники кодов операций. Программы стали более понятными для других. В начале 60-х годов XX в. разразился «кризис программирования», который был вызван несовершенством технологии программирования. Анализ причин позволил сформулировать новый подход к программированию.

Второй этап – структурный подход к программированию (60–70-е годы XX в.). Данный подход представляет собой совокупность рекомендуемых технологических приёмов, охватывающих выполнение всех этапов разработки программного обеспечения. Поддержка принципов структурного 5 программирования была заложена в основу так называемых процедурных языков программирования (Pascal, C и др.). В это время начала развиваться технология модульного программирования. Эту технологию поддерживают современные версии языков Pascal, C (C++). При увеличении размера программы возрастает сложность межмодульных интерфейсов. Для разработки программного обеспечения большого объёма было предложено использовать объектный подход.

Третий этап – объектный подход к программированию (с середины 80-х до конца 90-х годов XX в.). Объектно-ориентированное программирование определяется как технология разработки сложного программного обеспечения, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого типа (класса), а классы образуют иерархию с наследованием свойств. Взаимодействие программных объектов в такой системе осуществляется путём передачи сообщений (языки: Simula, Smalltalk, Pascal, C++, Modula, Java). Объектный подход предлагает новые способы организации программ, основанные на механизмах наследования, полиморфизма, композиции, наполнения. Но конкретная реализация в объектно-ориентированных языках программирования, таких как Pascal и C++, имеет существенные недостатки в части связи модулей. Связи модулей нельзя разорвать, но можно попробовать стандартизовать их взаимодействие.

Четвёртый этап – компонентный подход и CASE-технологии (с середины 90-х годов XX в. до настоящего времени). Компонентный подход предполагает построение программного продукта из отдельных компонентов физически отдельно существующих частей программного обеспечения, которые взаимодействуют между собой через стандартные двоичные интерфейсы. Это позволяет программистам разрабатывать продукты, хотя бы частично состоящие из повторно использованных частей. Компонентный подход лежит в основе технологий, разработанных на базе COM (Component Object Model – компонентная модель объектов), и технологии разработки распределённых приложений CORBA (Common Object Request Broker Architecture – общая архитектура с посредником обработки запросов объектов). Отличительной особенностью настоящего этапа развития технологии программирования, кроме изменения подхода, является разработка и внедрение автоматизированных технологий создания и сопровождения программного обеспечения, которые были названы CASE-технологиями (Computer-Aided Software / System Engineering – разработка программного обеспечения программных систем с использованием компьютерной поддержки).

Дальнейшая эволюция разработки Программного Обеспечения (ПО) привела к появлению **следующего этапа – no-code и low-code программирование.** При этом современное развитие индустрии нейросетей позволяет переложить часть задач генерации кода на нейросети, а готовые программы-конструкторы позволяют создавать работающие приложения без использования кодирования вообще.

1. Постановка задачи и спецификация программы

1.1. Постановка задачи – это процесс формулировки назначения программного обеспечения и основных требований к нему. Описываются *функциональные требования,* определяющие функции, которые должно выполнять программное обеспечение, и *эксплуатационные требования,* определяющие характеристики его функционирования. Этап постановки задачи заканчивается разработкой *технического задания (ТЗ)* с принятием основных проектных решений. Разработчик ТЗ называется технический писатель. ТЗ разрабатывает Исполнитель для Заказчика и формулируется в терминах, понятных Заказчику.

1.2. Техническое задание в полном объеме составляется в соответствии со стандартом ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению» [2] и имеет следующие основные разделы:

1.2.1. введение: наименование и краткая характеристика программного обеспечения;
1.2.2. основание для разработки;
1.2.3. назначение разработки: описание функционального и эксплуатационного назначения, спецификации функций;
1.2.4. требования к программному изделию: к функциональным характеристикам, к надежности, к техническим средствам;
1.2.5. требования к программной документации;
1.2.6. технико-экономические показатели;
1.2.7. технологические требования;
1.2.8. стадии и этапы разработки;
1.2.9. порядок контроля и приемки;
1.2.10. в техническое задание допускается включать приложения.

1.3. Технологические требования определяют выбор следующих принципиальных решений, влияющих на процесс проектирования программного обеспечения:

1.3.1. архитектура программного обеспечения;
1.3.2. пользовательский интерфейс;
1.3.3. метод программирования;
1.3.4. язык программирования;
1.3.5. среда программирования.

1.4. Архитектурой программного обеспечения называют **описание создаваемого программного обеспечения** на уровне его компонентов и связей между ними.

Архитектура программной системы во многом зависит от предметной области, для которой разрабатывается система. Поэтому часто архитектуры систем, разрабатываемых для одной и той же предметной области, имеют много общего. Следовательно, при проектировании архитектуры новой системы можно воспользоваться решениями, удачно примененными в ранее разработанных системах.

1.5. Развитие данного подхода привело к появлению ***архитектурных шаблонов*** **(паттернов),** на основе которых может создаваться архитектура конкретной системы. Архитектурный паттерн представляет собой описание типовой организации программной системы, опробованной в различных условиях и продемонстрировавшей свою эффективность.

Распространенными являются следующие архитектурные паттерны:

1.5.1. модель-представление-контроллер (*Model-View-Controller* (*MVC)*) применяется в системах, ориентированных на обслуживание клиентов;

1.5.2. паттерн хранилища данных применяется в системах, функционирование которых связано с обработкой большого объема данных (информационные системы, системы управления); архитектура таких систем должна содержат компоненты, генерирующие данные, и компоненты, их обрабатывающие;

1.5.3. паттерн «клиент-сервер» предусматривают распределение заданий между поставщиками и заказчиками услуг. Поставщики (серверы) предоставляют заказчикам (клиентам) определенный набор услуг (сервисов), доступ к которым осуществляется с помощью удаленного вызова соответствующих процедур;

1.5.4. паттерны потоков данных предполагают построение архитектуры системы из функциональных модулей, которые получают входные данные п преобразуют их в выходные. Преобразования могут осуществляться как последовательно, так и параллельно;

1.5.5. **многоуровневая система представляет собой иерархическую систему, состоящую из нескольких уровней, каждый из которых выполняет определенные функции. Каждый уровень предоставляет услуги вышестоящему уровню и использует услуги нижестоящего уровня.**

1.6. При процедурном программировании модель построения программы – иерархия множества подпрограмм, при **объектно-ориентированном программировании – иерархия множества классов**, совокупность обменивающихся сообщениями объектов классов. При этом простом виде архитектуры программа – это совокупность отдельно компилируемых программных единиц, используемая при решении задач. ***Пакеты программ*** – это совокупность программ, решающих задачи некоторой прикладной области. Например, пакет математических программ, пакет графических программ. ***Программные комплексы*** – это совокупность взаимосвязанных программ, совместно решающих небольшой класс задач некоторой прикладной области. Для решения задачи могут использоваться несколько программ комплекса, управляемые интерфейсом с пользователем, причем исходные данные и результаты желательно хранить в пределах одного проекта. ***Программные системы*** – это совокупность подсистем программ, совместно решающих большой класс сложных задач некоторой прикладной области. Отличие от программных комплексов – взаимодействие программ системы через общие данные и наличие развитых пользовательских интерфейсов.

1.7. Пользовательский интерфейс представляет собой совокупность программных и аппаратных средств, обеспечивающих диалог пользователя и компьютера. Различают следующие типы пользовательских интерфейсов:

1.7.1. примитивные интерфейсы (в том числе диалоговые интерфейсы);

1.7.2. **интерфейсы-меню (в студенческих программах);**

1.7.3. интерфейсы со свободной навигацией;

1.7.4. интерфейсы прямого манипулирования.

Разработка дизайна пользовательского интерфейса не входит в нашу задачу, т. к. имеются ограничения консольного режима. Однако при желании возможно использование соответствующих библиотек языков программирования для управления курсором в консольном режиме и построения более продвинутого интерфейса (**самостоятельно**).

1.8. К технологическим требованиям к программному обеспечению относятся:

1.8.1. выбор метода программирования: процедурный или объектно-ориентированный;

1.8.2. выбор языка программирования: **C++**, Java, Python и др.;

1.8.3. выбор среды программирования: **Visual Studio** фирмы Microsoft, Embarcadero RAD Studio (включающая Delphi и C++ Builder), Eclipse и др.

1.9. Спецификациями называют полное и точное описание функций и ограничений разрабатываемого программного обеспечения. Существуют **функциональные спецификации**, описывающие функции разрабатываемого программного обеспечения, и эксплуатационные спецификации, описывающие требования к техническим средствам. Требование полноты означает строгое соответствие техническому заданию, а требование точности – однозначное толкование спецификаций разработчиком и заказчиком.

Функциональные спецификации содержат:

1.9.1. декомпозицию и содержательную постановку задач;

1.9.2. эксплуатационные ограничения;

1.9.3. математические методы решения;

1.9.4. модели программного обеспечения.

2. Модели программного обеспечения

Модели программного обеспечения зависят от технологии программирования (процедурная или объектно-ориентированная).

2.1. При **процедурном** программировании используются следующие модели разрабатываемого программного обеспечения:

2.1.1. **функциональные диаграммы, отражающие взаимосвязи функций разрабатываемого программного обеспечения и ориентированные на иерархию функций, декомпозицию функций**;

2.1.2. диаграммы потоков данных, описывающие взаимодействие источников и потребителей информации и позволяющие специфицировать как функции, так и данные;

2.1.3. диаграммы структур данных, описывающие базы данных разрабатываемого программного обеспечения;

2.1.4. диаграммы переходов состояний, описывающие поведение разрабатываемого программного обеспечения при получении управляющих данных извне (например, команды пользователя).

2.2. При объектно-ориентированном программировании модели разрабатываемого программного обеспечения основаны на предметах реального мира. **На этапе определения спецификаций требуется разработать модель предметной области программного обеспечения на базе иерархии классов (типов, определенных пользователем), объектной декомпозиции.** Разрабатываемое программное обеспечение представляется в виде совокупности объектов (экземпляров классов), в результате взаимодействия которых через передачу сообщений происходит выполнение требуемых функций. Графическое отображение модели ПО в этом случае представляется в виде диаграмм на языке UML (Unified Modeling Language). Унифицированный язык моделирования (UML) – это семейство графических нотаций, в основе которого лежит единая метамодель. Он помогает в описании и проектировании программных систем, в особенности систем, построенных с использованием объектно-ориентированных (ОО) технологий. Это определение в чем-то упрощенное. В действительности разные люди могут видеть в UML разные вещи. Это является следствием как собственной истории развития языка, так и различных точек зрения специалистов на то, что делает процесс разработки программного обеспечения эффективным. Графические языки моделирования уже продолжительное время широко используются в программной индустрии. Основная причина их появления состоит в том, что языки программирования не обеспечивают нужный уровень абстракции, способный облегчить процесс проектирования. Несмотря на то что графические языки моделирования существуют уже достаточно давно, в среде разработчиков программного обеспечения очень много спорят об их роли. Эти споры оказывают непосредственное влияние на восприятие разработчиками самого языка UML. UML представляет собой относительно открытый стандарт, находящийся под управлением группы OMG (Object Management Group – группа управления объектами), открытого консорциума компаний. Группа OMG была сформирована для создания стандартов, поддерживающих межсистемное взаимодействие, в частности взаимодействие объектно-ориентированных систем. Группа OMG более известна по стандартам CORBA (Common Object Request Broker Architecture – общая архитектура посредников запросов к объектам). UML появился в результате процесса унификации множества объектноориентированных языков графического

моделирования, процветавших в конце 80-х и в начале 90-х годов. Более подробно изучить язык UML можно в [8].

3. Жизненный цикл (ЖЦ) программы

Совокупность процессов от момента появления идеи создания ПО до окончания его эксплуатации.

Структура ЖЦ ПО: ГОСТ Р ИСО/МЭК 12207-2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств».

3.1. Архитектура ЖЦ ПО.

3.1.1. Основные процессы:

3.1.1.1. Приобретение.

3.1.1.2. Поставка.

3.1.1.3. **Разработка ПО**.

3.1.1.4. Эксплуатация.

3.1.1.5. Сопровождение.

3.1.2. Поддержка:

3.1.2.1. Документирование.

3.1.2.2. Конфигурационное управление.

3.1.2.3. Обеспечение качества.

3.1.2.4. Верификация.

3.1.2.5. Валидация (проверка правильности).

3.1.2.6. Управление проектом.

3.1.2.7. Ревизия отчетов.

3.1.2.8. Устранение дефектов.

3.2. Процесс **Разработка ПО (жирным выделены процессы, включенные в студенческие работы)** включает в себя:

3.2.1. Подготовка процесса (выбор модели ЖЦ, стандартов, плана работ).

3.2.2. **Анализ требований к системе (функциональных и пользовательских-ТЗ)**.

3.2.3. **Проектирование архитектуры системы**.

3.2.4. **Анализ требований к ПО (определение возможностей и внешних интерфейсов-спецификации)**.

3.2.5. Проектирование архитектуры в т. ч. **интерфейсов**.

3.2.6. Детальное проектирование ПО (**описание компонентов и интерфейсов между ними, требования к тестовым примерам)**.

3.2.7. **Кодирование и тестирование компонентов ПО**.

3.2.8. **Интеграция компонентов**.

3.2.9. **Квалифицированное тестирование (в присутствии заказчика)**.

3.2.10. **Интеграция системы**.

3.2.11. **Аттестационное тестирование с проверкой документации**.

3.2.12. Инсталляция ПО.

3.2.13. **Приемка ПО**.

3.3. Укрупненные этапы разработки ПО согласно ГОСТ 19.102-77 «Стадии разработки» [3].

Используем более упрощенную схему по сравнению с ГОСТ [2], т. к. экономические показатели, например, не входят в предмет изучения.

3.3.1. **Постановка задачи (стадия ТЗ –** для заказчика**)**.

3.3.2. **Определение спецификаций (стадия «Эскизный проект»**.

3.3.3. **Проектирование (стадия «Технический проект»)**.

3.3.4. **Реализация (стадия «рабочий проект»)**.

3.3.5. *Сопровождение (может не быть)*.

4. Виды разработки программных средств

4.1. Авторская разработка (как правило, создание наукоемких приложений, узкая специализация).

4.2. Коллективная разработка (Разделение труда между работниками). Модели см. ниже.

Рис. 1. Иерархическая модель разработки ПО

Рис. 2. Матричная модель разработки ПО

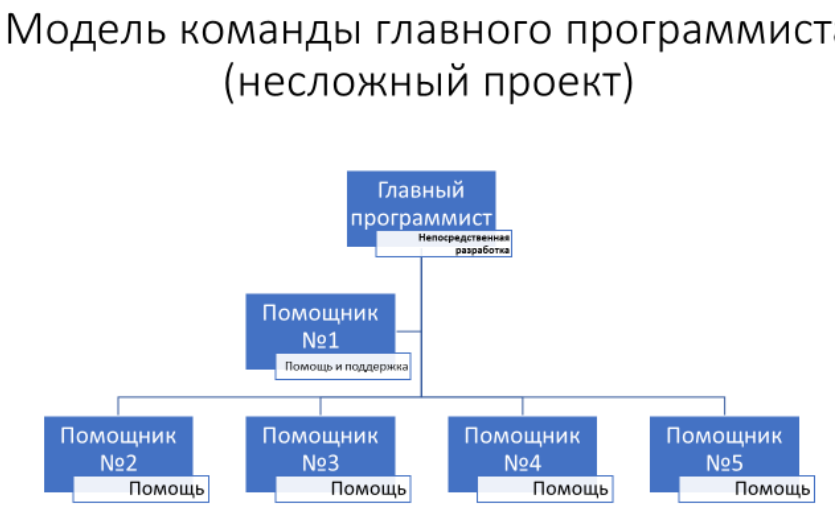


Рис. 3. Модель команды главного программиста

Рис. 4. Ядерная модель разработки ПО

5. Выполнение лабораторных работ

5.1. При изучении дисциплины «Технологии и методы программирования» студенты выполняют лабораторные работы 1 и 2 **в виде деловой игры**, имитирующей технологический процесс (бизнес-процесс) создания программного продукта в коллективе малого предприятия с оформлением документооборота между **Заказчиком и Исполнителем, Начальником и подчиненными** на основе созданных студентами в предыдущем семестре программ. Взаимоотношения между Заказчиком и Исполнителем отображаются только созданием ТЗ. Маркетинговые и бухгалтерские документы не являются темой данного пособия. В роли Заказчика выступает преподаватель, в роли главного разработчика (начальника) или директора фирмы Исполнителя выступает студент, создающий прототип, комплект документов для разработки, основную программу и отчет. В роли удаленного программиста-кодировщика выступает напарник по группе. Аналогично для создания другого программного продукта напарники меняются местами. На выполнение лабораторной работы № 1 отводится один календарный месяц, согласно предоставленному разработчиком ПО календарному плану работ. Работа № 2 служит для закрепления материала и не требует напарника в качестве удаленного программиста.

Модель данного бизнес-процесса выглядит следующим образом:

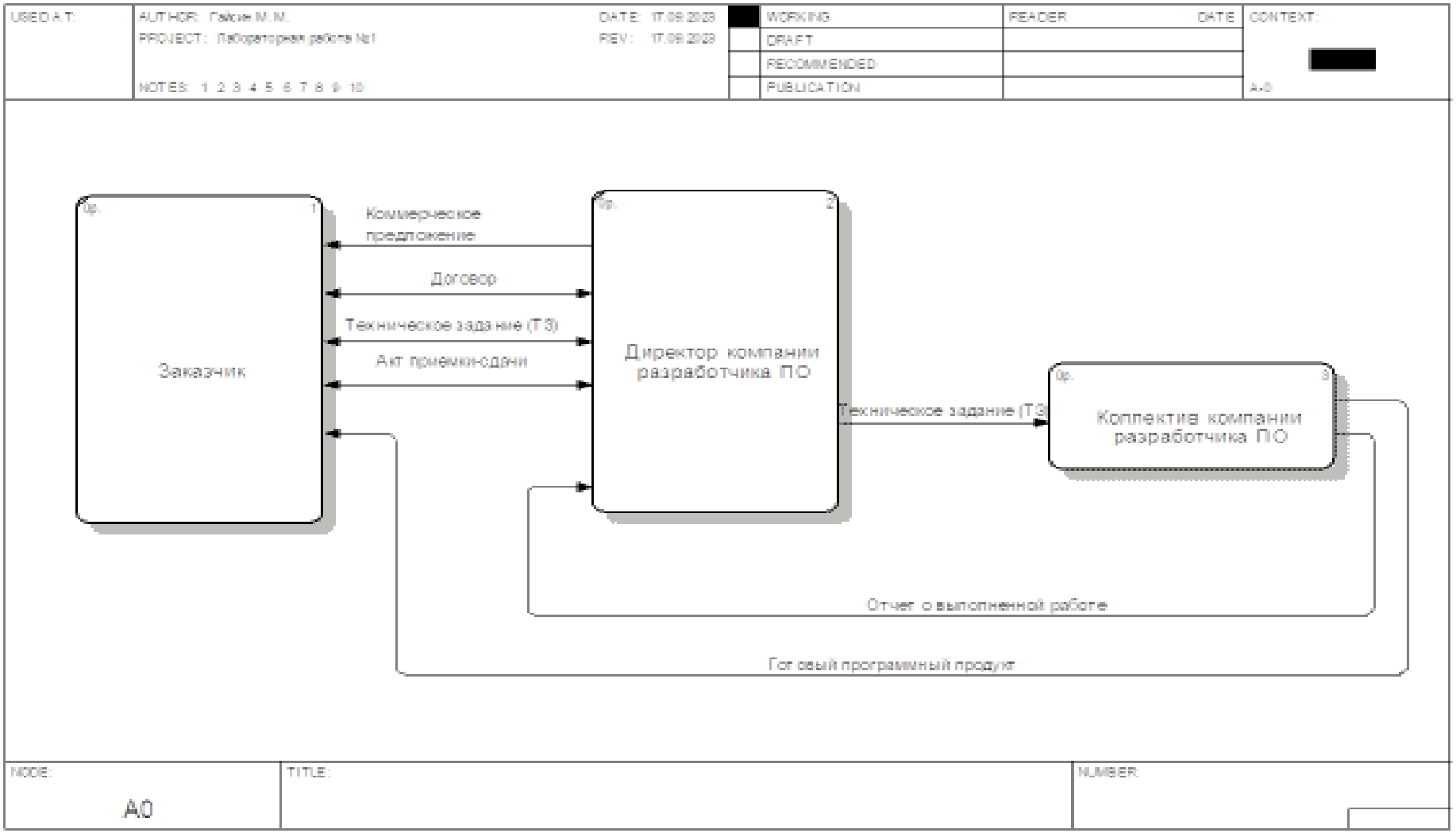


Рис. 5. Взаимодействие между Заказчиком и Исполнителем

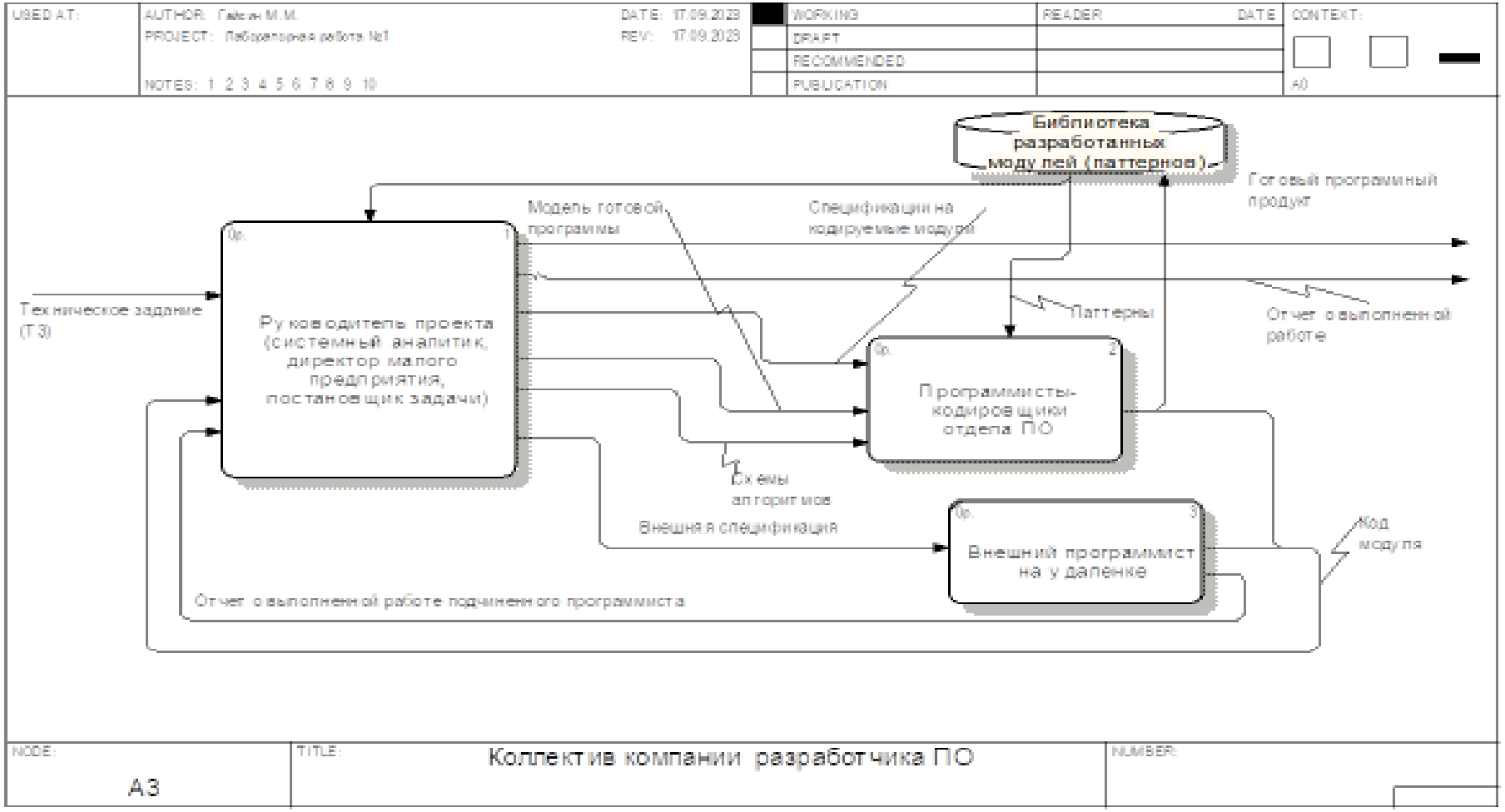


Рис. 6. Документооборот внутри компании Разработчика

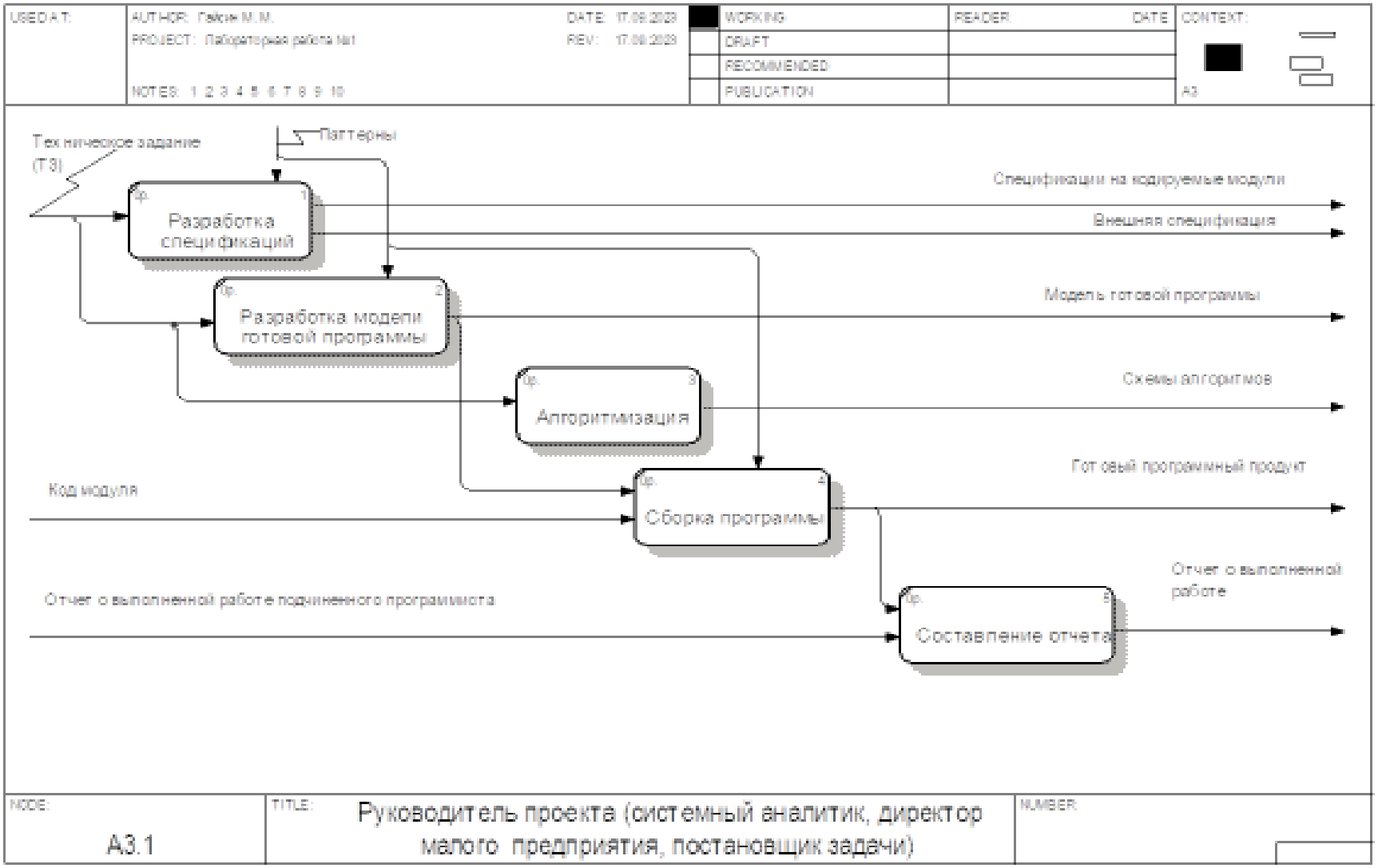


Рис. 7. Функции руководителя

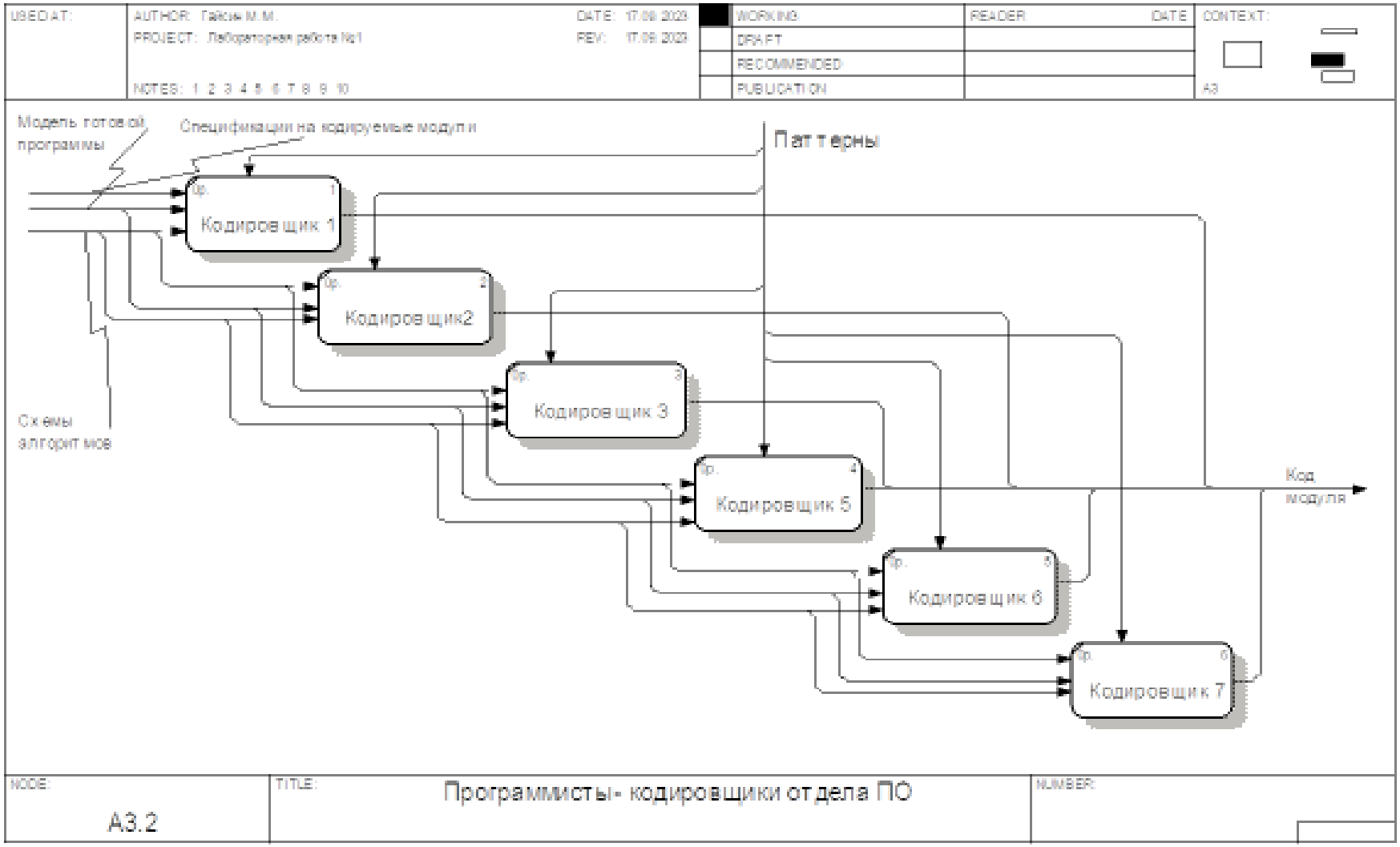


Рис. 8. Функции кодировщиков

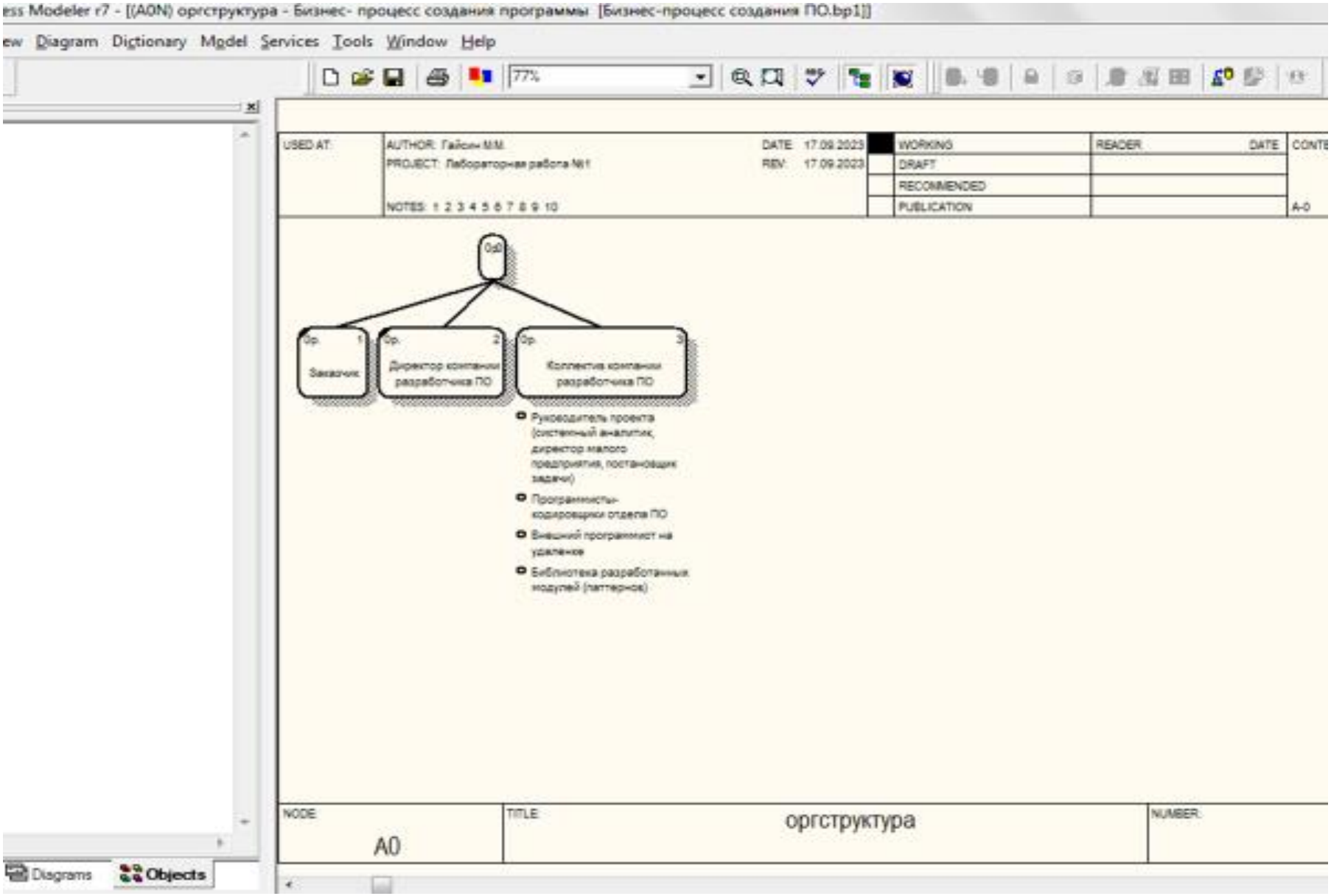


Рис. 9. Организационная структура проекта

Таким образом студентами реализуется **ядерная модель** создания программного продукта (см. рис. 4).

5.2. Варианты заданий по лабораторной работе № 1 – это лабораторные работы № 1–6 из предыдущего семестра плюс задание, которое начальник выдает удаленному подчиненному в виде внешней спецификации (варианты заданий из глав 3–5 Самоучителя [1] на усмотрение начальника по согласованию с преподавателем). На основе этого составляется ТЗ для условного заказчика и спецификации для программистов-кодировщиков (в том числе и внешняя спецификация для подчиненного (удаленного программиста). **Задача студента**: на основе им созданных документов: ТЗ, спецификаций, программ разработанных в прошлом семестре и программы, которую на основании внешней спецификации для него напишет подчиненный программист-кодировщик, создать программу, решающую все указанные задачи с возможностью выбора любой задачи в пунктах меню интерфейса. В программе обязательно должны быть предусмотрены:

1) возможности многократного решения задач;
2) ввода-вывода данных из/в файлы, в случае большого объема вводимых данных предусмотреть генерацию случайных чисел;
3) защита от несанкционированных действий пользователя;
4) контрольные примеры решения задач, иллюстрирующие правильность исполняемых алгоритмов (допустимо контрольные примеры размещать в образцах входных файлов);
5) унификация всех видов меню всех подпрограмм (например, 0-выход в предыдущее меню, 1-продолжение работы);
6) оптимизация кодов подпрограмм, при этом защита от некорректных действий пользователя и работа с файлами выносятся в отдельные модули, используемые как при решении текущих задач, так и в следующих лабораторных работах.

Программы пишутся на языке C++ и после отладки предоставляются преподавателю. После устранения всех замечаний по документации и программе составляется отчет (требования к отчету см. ниже) и сдается преподавателю без ошибок в электронном виде в формате MS Word.

6. Цель работы

Цель работы, согласно данных методических рекомендаций, − привить студентам навыки составления технологического процесса создания программного продукта в коллективе малого предприятия, взаимодействия начальник – подчиненный, унификации программного обеспечения и оформления документооборота согласно ГОСТам и требованиям ЕСПД.

7. Перечень документов и требования к отчету по лабораторной работе № 1

7.1. Техническое задание

Техническое задание по ГОСТ 19.102-77 [3] включает в себя следующие работы:

7.1.1. Постановка задачи *(включает общую постановку задачи и постановку задач по каждому отдельному пункту меню)*.
7.1.2. Сбор исходных материалов *(отчеты за прошлый семестр и Самоучитель).*
7.1.3. Выбор и обоснование критериев эффективности и качества разрабатываемой программы (*см. ниже*).
7.1.4. Обоснование необходимости проведения научно-исследовательских работ (*не требуются, но пункт обязателен)*.

7.1.5. Определение структуры входных и выходных данных *(что Заказчик будет вводить и что получит в результате в терминах специалистов Заказчика).*

7.1.6. Предварительный выбор методов решения *задач (вся математика задач из прошлого семестра)*.

7.1.7. Обоснование целесообразности применения ранее разработанных программ *(программы превращаем в подпрограммы, модернизируем, оптимизируем)*.

7.1.8. Определение требований к техническим средствам *(компьютер, система, прикладной софт)*.

7.1.9. Обоснование принципиальной возможности решения поставленной задачи.

7.1.10. Разработка и утверждение технического задания.

7.1.10.1. Определение требований к программе *(относительно возможных ошибок, язык интерфейса, защита от некорректных действий пользователей и т. п.).*

7.1.10.2. Разработка технико-экономического обоснования разработки программы (*не требуются, но пункт обязателен)*.

7.1.10.3. Определение стадий, этапов и сроков разработки программы и документации на нее *(расписать по этапам один календарный месяц)*.

7.1.10.4. Выбор языков программирования.

7.1.10.5. Согласование и утверждение технического задания.

П. 7.1.3. Выбор и обоснование критериев эффективности и качества разрабатываемой программы включает в себя:

7.1.3.1. Мобильность *(независимость от среды)*.

7.1.3.2. Надежность *(устойчивость работы, точность)*.

7.1.3.3. Эффективность *(минимальные расходы вычислительных ресурсов).*

7.1.3.4. Дружественность *(дружественный интерфейс)*.

7.1.3.5. Модифицируемость.

7.1.3.6. Коммуникативность *(интеграция с др. программами и возможность обмена данными с ними)*.

Все данные пункты требуют расшифровки относительно разрабатываемой программы.

7.2. Спецификация

Спецификация пишется по основной программе и каждому модулю. Условно считаем, что каждый модуль пишет отдельный программист-кодировщик. Текст пишется в повелительном наклонении. Спецификация – фактически приказ программисту на кодирование программы согласно словарю алгоритмического языка программирования. Спецификация вторична и составляется на основе ТЗ опытным программистом, системным аналитиком или руководителем проекта после подписания Заказчиком ТЗ. При наличии у Заказчика своих специалистов указанного профиля спецификации могут составляться совместно с ними и также подписываться Заказчиком. Формально для кодировщика необходимы два документа: Спецификация и блок-схема алгоритма программы. Блок-схемы модулей программы уже разработаны в прошлом семестре и дополнительной разработки не требуют.

7.2.1. Спецификация включает в себя:

7.2.1.1. постановку задачи *(на основании ТЗ постановка пишется в терминах программиста и для программиста, возможны гиперссылки на ТЗ);*

7.2.1.2. анализ этой задачи;

7.2.1.3. подробное описание действий, которые должна выполнять программа.

7.2.2. В спецификации отражаются:

7.2.2.1. состав входных, выходных и промежуточных данных *(конкретные типы и названия переменных, констант, массивов и пр., с указанием их назначения, входные и выходные файлы);*

7.2.2.2. какие входные данные являются корректными и какие ошибочными;

7.2.2.3. кто является пользователем программы и каким должен быть интерфейс *(детальное описание интерфейса пользователя со всеми вариантами и описание программного интерфейса (API) – между модулями и функциями);*

7.2.2.4. какие ошибки должны выявляться и какие сообщения должны выдаваться пользователю *(весь перечень ошибочных ситуаций и реакции программы на них);*

7.2.2.5. какие ограничения имеет программа (например, программа расчета факториала может иметь ограничение по максимальному числу);

7.2.2.6. все особые ситуации, которые требуют специального рассмотрения;

7.2.2.7. какая документация должна быть подготовлена;

7.2.2.8. перспективы развития программы.

В спецификации обязательно учитывается применение готовых модулей защиты от некорректных действий пользователя и работы с файлами. На документы и файлы данных обязательно наличие корректных гиперссылок.

7.3. Схема основного алгоритма

Схема основного алгоритма (схемы алгоритмов модулей задач прошлого семестра не требуются, кроме № 7). Файловый ввод-вывод входит в схему основного алгоритма, все программы должны быть модернизированы таким образом, чтобы файловый ввод-вывод происходил путем обращения к одному перегруженному модулю (допустимо разделять ввод и вывод в разных модулях на усмотрение разработчика).

7.4. Внешняя спецификация

Внешняя спецификация для подчиненного программиста на удаленке (модуль № 7). Содержание см. выше.

7.5. Пример внешней спецификации

Пример внешней спецификации *(текст и стиль студенческой работы сохранен)*.

7.3.1. Постановка задачи: написать программу для реализации кодирования текста на основе базовой строки. Текст и ключ для кодирования должны быть реализованы в виде строковых динамических массивов. Программа должна запрашивать исходный текст и ключ, а также позволять ввод данных из файла. Вывод закодированного текста должен быть осуществлен на экран и записывается в файл. Программа должна обрабатывать возможные ошибки ввода и позволять пользователю выбирать между вводом данных вручную или чтением из файла.

7.3.2. В подпрограмме использовать вспомогательные функции (см. Спецификацию вспомогательных функций – может входить в общую спецификацию или представлять из себя отдельный документ):

7.3.3. для вывода описания подпрограммы showFileInfo со значением аргумента “./config/info/sub7.txt”;

7.3.4. для вывода меню и получения значения выбранного пункта меню selectedOption с аргументом “./config/menu/sub.txt”;

7.3.5. для чтения данных из файла inputFromFile;

7.3.6. для записи результата в файл saveToFile.

7.3.7. Добавить функцию от неправильного ввода значений (try-catch (с файла), проверку корректности значений (базовая проверка типов присутствует в функции manualEntry используется values = manualEntry(values, "Введите строку для кодирования: ", 1); // Ввод одной строки.

7.3.8. values = manualEntry(values, "Введите ключ: ", 1); // Ввод ключа).

7.3.9. Подпрограмму необходимо реализовать в виде c++ модуля с расширением «.hpp», модуль и головную функцию подпрограммы необходимо назвать «sub7».

7.3.10. В головной функции подпрограммы необходимо реализовать:

зацикливание и выход из нее через меню по желанию пользователя.

7.3.11. Состав входных, выходных и промежуточных данных

7.3.12. **Входные данные**:

7.3.12.1. **Строка для кодирования** (вводится пользователем либо считывается из файла(examples\7.txt)).

7.3.12.2. **Ключ** (вводится пользователем либо считывается из файла).

7.3.12.3. В коде это переменные values [0] (строка для кодирования) и values [1] (ключ).

7.3.13. **Промежуточные данные**:

7.3.13.1. **Закодированная строка** (переменная encrypted), которая генерируется функцией encryptText на основе исходной строки и ключа.

7.3.13.2. **Переменная** string temp, // temp для временных значений; которая используется для временного хранения введённых данных (строки и ключа).

7.3.13.3. **Меню выбора** (menu), где пользователь выбирает действие (например, ввод данных вручную или чтение из файла).

7.3.14. **Выходные данные**:

7.3.14.1. **Закодированный текст**, который выводится на экран с помощью cout и сохраняется в файл через функцию saveToFile.

7.3.14.2. Сообщение о завершении операции кодирования (например, вывод закодированного текста).

7.3.15. **Корректные и ошибочные входные данные.**

7.3.16. Корректные входные данные:

7.3.16.1. Корректные данные – это те, которые соответствуют ожиданиям программы и позволяют ей успешно выполнить поставленную задачу.

7.3.16.2. **Строка для кодирования** (параметр text):

7.3.16.3. Корректная строка должна содержать символы, которые программа сможет обработать (например, текст на английском или русском языке).

7.3.16.4. Строка должна быть не пустой.

7.3.16.5. **Ключ для шифрования** (параметр key):

7.3.16.6. Ключ должен быть строкой произвольной длины, но его длина должна быть больше нуля, иначе программа не сможет корректно его применить в цикле шифрования.

7.3.16.7. Ключ может состоять из букв, цифр или специальных символов.

7.3.16.8. Пример корректных входных данных:

7.3.16.8.1. Строка: "HelloWorld".

7.3.16.8.2. Ключ: "Key123".

7.3.17. Ошибочные входные данные.

Ошибочные данные – это данные, которые программа не может корректно обработать, что может привести к некорректной работе, ошибкам или завершению программы.

7.3.18. **Примеры ошибок:**

7.3.18.1. **Пустая строка для кодирования**: программа ожидает, что строка будет содержать хотя бы один символ. Если строка пуста, результат шифрования будет некорректным или пустым. Например, если ввести пустую строку, то программа ничего не зашифрует.

7.3.18.2. **Пустой ключ**: если ключ пустой, то цикл шифрования не сможет корректно работать, так как он основан на повторении символов ключа. Это может привести к некорректной работе программы

7.3.18.3. **Ввод некорректных символов**: если в строке для кодирования или в ключе будут символы, которые программа не поддерживает (например, символы с кодами вне диапазона ASCII или UTF-8), это может привести к ошибкам при шифровании.

7.3.18.4. Пример ошибочных данных:

7.3.18.4.1. Строка: ""

7.3.18.4.2. Ключ: ""

7.3.19. **Пользователь программы и интерфейс.**

7.3.20. Пользователь программы:

7.3.20.1. студенты, изучающие кодирование текста;

7.3.20.2. преподаватели, которые используют программу для демонстрации кодирование текста на основе базовой строки;

7.3.20.3. разработчики, желающие использовать функционал программы в своих проектах.

7.3.21. Интерфейс программы:

7.3.22. Программа должна быть реализована с использованием текстового меню, которое предлагает пользователю несколько опций для взаимодействия. Основное взаимодействие происходит через выбор пунктов меню с возможностью ввода данных и работы с файлами. Каждый пункт меню соответствует определённой задаче.

7.3.23. Структура меню программы:

7.3.24. 1. Главное меню:

7.3.25. Задача № 7: Вводится пользователю, чтобы начать работу с программой. Заголовок задачи должен содержать сведения о разработчике.

7.3.26. Программа должна выводить основное меню для выбора дальнейших действий.

7.3.27. **Пункты меню:**

7.3.28. 1. Ввод значений из файла:

7.3.29. В этом режиме программа запрашивает файл, из которого считываются исходный текст и ключ для кодирования. Если происходит ошибка при чтении файла (например, файл не существует или имеет неверный формат), программа выведет сообщение об ошибке и предложит повторить ввод.

7.3.30. 2. Ввод значений вручную:

Программа предлагает пользователю ввести вручную исходный текст и ключ для кодирования.

После ввода данных они сохраняются в переменные для дальнейшей обработки. Этот пункт позволяет работать с программой без использования файлов.

7.3.31. 0. Выход из программы:

Позволяет пользователю выйти из программы. При выборе этого пункта программа должна завершать свою работу. При работе в составе головной программы начальника по данному пункту должен происходить переход в головное меню.

3. Дополнительные действия:

– Программа должна предоставлять возможность просмотра закодированного текста сразу на экране после кодирования.
– Пользователь может сохранить закодированный текст в файл. Программа должна сохранять как исходный текст и ключ, так и результат кодирования.

7.3.32. **Логика работы:**

7.3.33. После выбора нужного пункта меню, программа либо должна запрашивать данные для кодирования, либо считывать их из файла. Кодирование должно происходить на основе функции, которая использует базовую строку (ключ) для шифрования. После выполнения операции программа должна отображать результат на экране и предлагать сохранить его в файл.

7.3.34. **Ошибки и их сообщения**: если menu не число или не равно 0, 1 или 2, вывести: «выбранный пункт меню отсутствует».

7.3.35. **Все особые ситуации, которые требуют специального рассмотрения**:

– нет.

7.3.36. **Ограничения программы**:

– нет.

7.3.37. Документация:

7.3.38. Отчет подчиненного, включающий в себя:

7.3.38.1. Титульный лист.

7.3.38.2. Содержание.

7.3.38.3. Цель работы.

7.3.38.4. Схему алгоритма.

7.3.38.5. Скриншоты (основное меню и каждый модуль с вариантами работы с защитой от несанкционированных действий пользователя).

7.3.38.6. Код программы в двух видах: общая программа с main для проверки и модуль (модули) для встраивания в программу начальника.

7.3.38.7. Вывод о соответствии разработанной программы требованиям Внешней спецификации.

7.3.39. Отчет подчиненного составляется по образцу прошлого семестра (начальник отвечает за подчиненного, подчиненный составляет программу таким образом, чтоб начальник мог ее скопировать и вставить в свою программу без какой-либо модификации. Принимает отчет начальник).

7.6. Перспективы развития программы

Перспективы развития программы: программа может быть доработана по желанию пользователя.

7.7. Текст программы

Текст программы (*подробнее см. дальше*).

7.8. Отчет о проделанной работе

Отчет о проделанной работе (может быть представлен как в виде одного файла, так и в виде пакета документов с гиперссылками на каждый документ). Рубрикация может быть как сквозная по всем документам, так и отдельная. Если документ содержит более 2 страниц, то в документе должна быть страница Содержание Отчет – основной документ и включает в себя:

7.8.1. Титульный лист (оформленный в соответствии с предъявляемыми требованиями).

7.8.2. Цель работы (своя для каждой лабораторной работы).

7.8.3. Задание.

7.8.4. Содержательную часть (см. выше) или гиперссылки на соответствующие документы.

7.8.5. Вывод о результатах выполненной работы.

7.9. Примеры выполнения с замечаниями

Примеры выполнения с замечаниями (***курсивом***).

7.9.1. Техническое задание (*рубрикация работы студента сохранена!*).

Оглавление

Техническое задание

1. Обоснование необходимости разработки программы: учебный процесс.

1.1. Постановка задачи.

Написать программу, которая должна решать 7 задач. Программа должна отвечать всем требованиям защиты от ввода некорректных данных пользователем, быть зациклена, предоставлять возможность ввода данных с консоли и с файла, а также выводить результат на экран и в файл *(не отмечено, что для больших объемов данных обеспечить генерацию случайных чисел)*.

1.1.1. Задача 1

Задача 18. Написать программу для вычисления кинетической энергии E_k релятивистского электрона. Использовать формулу

$$E_k = \frac{m_e c^2}{\sqrt{1 - \left(v/c\right)^2}} - m_e c^2,$$

где масса (масса покоя) электрона $m_e \approx 9.1 \times 10^{-31}$ (кг), скорость света $c \approx 2.998 \times 10^8$ (м/с), а v – скорость электрона. Для сравнения рассчитайте нерелятивистское выражение $E = \frac{m_e v^2}{2}$.

1.1.2. Задача 2

Задача 16. Написать программу для вычисления методом последовательных итераций уравнения $x = A\exp(-x)$. Параметр A вводится пользователем. Проверить, для каких значений параметра A применим метод последовательных итераций.

Верхняя граница произведения вводится пользователем с клавиатуры.

1.1.3. Задача 3

Задача 9. Моментом порядка k случайной величины ξ называется число $M\xi^k = \sum_{i=1}^{n} \xi_i^k p_i$, где случайная величина ξ принимает значения ξ_1, ξ_2, ..., ξ_n с вероятностями p_1, p_2, ..., p_n соответственно. Написать программу для вычисления момента порядка k для случайной величины с заданным законом распределения. Порядок момента и закон распределения вводятся пользователем.

1.1.4. Задача 4

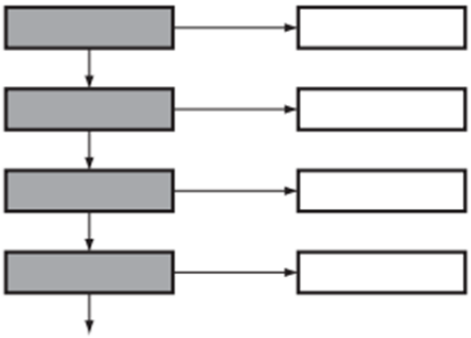
Задача 12. Написать программу с функцией для поиска экстремального (наибольшего или наименьшего) элемента массива. Массив заполнить случайными числами.

1.1.5. Задача 5

Задача 12. Написать программу для транспонирования матрицы, реализованной в виде двумерного динамического массива.

1.1.6. Задача 6

Написать программу для создания динамической конструкции из элементов двух типов (Список с ветвлением). Элементы первого типа ссылаются на два элемента. Элементы второго типа не содержат полей для выполнения ссылок. Элементы первого типа образуют последовательный список. Каждый из элементов этого списка ссылается на следующий элемент и на элемент второго типа.



1.1.7. Задача 7

Задача 1. Написать программу с функцией для вычисления числа из последовательности Фибоначчи. Аргументом функции является порядковый номер числа в последовательности.

1.2. Исходные материалы:

Отчёты по лабораторным работам 1–6, студента Института права, социального управления и безопасности УдГУ ФИО, 2 курса ИБ ОБ-10.03.01.02-21.

Самоучитель С++ с примерами. 4-е издание (переработанное). Книга + виртуальный CD. – СПБ. : Наука и Техника, 2016. – 480 с.: ил. (+ виртуальный CD).

Отчёт по работе подчинённого программиста студента Института права, социального управления и безопасности УдГУ ФИО, 2 курса ИБ ОБ-10.03.01.02-21.

1.3. Выбор и обоснование критериев эффективности и качества разрабатываемой программы. *Недостаточно критериев (см. выше).*

Дружественность. Программа должна быть понятна для пользователя, т. е. интерфейс программы подразумевает описание всех использованных терминов на русском языке для русскоязычного пользователя; программа должна быть удобна для пользователя, т. е. интерфейс должен быть представлен в виде меню и диалога.

Надежность. В программе должна быть предусмотрена защита от некорректных действий пользователя; программа не должна содержать критических ошибок, влияющих на работоспособность программы.

Коммуникативность. Программа должна работать с библиотеками для корректной работы программы; должна быть предусмотрена возможность коммуникации с внешними программами посредством текстовых файлов.

1.4. Обоснования необходимости проведения научно-исследовательских работ.

– Не требуется.

1.5. Определение структуры входных и выходных данных.

1.5.1. Задача 1

Входные: v – значение скорости электрона в м/с.

Выходные: Ek – кинетическая энергия релятивистского электрона, E – значение нерелятивистского движения.

1.5.2. Задача 2

Входные: число A – значение параметра для данного в 1.1.2 уравнения.

Выходные: x – корень уравнения, согласно формуле из пункта 1.1.2.

1.5.3. Задача 3

Входные: n – целое число (количество реализуемых значений), k – целое число (порядок момента), xi [101] – массив дробных чисел (значения случайной величины), p [101] – массив дробных чисел (значения вероятности).

Выходные: Mxik – значение момента порядка k случайной величины xi по формуле из пункта 1.1.3.

1.5.4. Задача 4

Входные: size – целое число (размер массива).

Выходные: max – вещественное число (максимальный элемент массива), min – вещественное число (минимальный элемент массива).

1.5.5. Задача 5

Входные: n – целое число (строки матрицы) , m – целое число (столбцы матрицы), nm [n][m] – массив дробных чисел (элементы исходной матрицы).

Выходные: nm [m][n] – массив дробных чисел (элементы транспонированной матрицы).

1.5.6. Задача 6

Входные: n – целое число (количество элементов в списке), значения для элементов первого и второго типов (вещественные числа).

Выходные: список элементов в формате, показанном на схеме в п. 1.1.6.

1.5.7. Задача 7

Входные: целое число n – порядковый номер числа в последовательности Фибоначчи.

Выходные: Вычисленное целое число Фибоначчи при порядковом числе n.

Недостаток: не указаны примеры файлов входных и выходных данных.

1.6. Предварительный выбор методов решения задач.

Вычисление значений кинетической энергии релятивистского электрона и нерелятивистского выражения согласно формулам, данным в п. 1.1.1.

Вычисление методом последовательных итераций уравнения, данного в п. 1.1.2.

Сначала выбираем начальное значение для переменной, с которого начнём вычисления. Далее создаём цикл, где будем повторять процесс: на каждом шаге считаем новое значение

переменной по заданной формуле и заменяем старое значение на новое. Цикл продолжается, пока разница между старым и новым значением не станет очень маленькой или пока не будет выполнено нужное количество шагов. В конце выводим результат – решение уравнения.

Нахождение момента случайной величины по формуле, данной в пункте 1.1.3.

Нахождение экстремального элемента массива:

Берем первый элемент как максимум и минимум; сравниваем каждый следующий элемент: обновляем максимум и минимум при необходимости (если встречается число больше/меньше текущего); в конце получаем самые большие и маленькие значения массива.

Транспонирование матрицы:

Меняем местами строки и столбцы: то, что было в строках, переносим в столбцы, и наоборот; для каждой строки записываем значения по столбцам в новую строку; в результате получаем новую матрицу, где строки стали столбцами, а столбцы – строками.

Формирование списка с ветвлением: *нумерация сквозная (отдельную со скобками делать недопустимо!):*

1) формируем список, где каждый элемент состоит из двух типов данных: основной и дополнительный;
2) каждый элемент первого типа хранит ссылку на элемент второго типа;
3) связываем элементы первого типа в один список.

В итоге получается структура, где у каждого элемента списка есть собственное значение и связанный с ним дополнительный элемент.

1.6.7. Вычисление числа Фибоначчи:

1) начальные условия: начинать с двух первых чисел последовательности: 0 и 1;
2) рекуррентное соотношение: для каждого следующего числа в последовательности использовать формулу F(n) = F(n-1) + F(n-2). Это означает, что нужно сложить два предыдущих числа, чтобы получить следующее;
3) итерация: продолжать процесс, пока не будет достигнуто нужное число (согласно порядковому номеру, введенному пользователем) в последовательности.

1.7. Применение ранее разработанных программ целесообразно, для этого:

– необходимо добавить защиту от ввода некорректных данных, введенных пользователем согласно пункту 1.3., добавить возможность чтения данных из файла и записи результата в файл в программы 1–2;

– объединить 7 подпрограмм в одну единую программу с понятным и доступным интерфейсом, структурировать её;

– предоставить возможность пользователю повторного выполнения программы.

1.8. Определение требований к техническим средствам.

На компьютере должна быть установлена операционная система Windows (начиная с версии 7 и более) и компилятор Visual Studio 2022.

1.9. Обоснование принципиальной возможности решения поставленной задачи.

Решение поставленной задачи возможно, так как при написании программы будет использован универсальный язык программирования C++. Команда разработчиков также имеет опыт решения подобных задач.

2. Разработка и утверждение технического задания.

2.1. Определение требования к программе:

– программа должна быть понятна конечному пользователю;

– программа должна выводить все сообщения на русском языке для русскоговорящего пользователя;

– программа должна предупреждать пользователя о некорректном вводе во избежание ошибок (пункт 1.3.1.);

– программа должна использоваться в соответствии с типовым лицензионным соглашением для программных продуктов аналогичного вида;

– программа не должна содержать критических ошибок, влияющих на работу самой программы;

– программа не должна предполагать работы с большими объемами информации.

2.2. Разработка технико-экономического обоснования разработки программы.

– Не требуется.

2.3. Определение стадий, этапов и сроков разработки программы и документации на неё.

Техническое задание (20.09.24), спецификация (23.09.24), внешняя спецификация (25.09.24), отчёт подчинённого (27.09.24), схема основного алгоритма (30.09.24), текст программы и скриншоты (30.09.24), отчёт о проделанной работе (30.09.24).

2.4. Выбор языков программирования.

Для реализации задачи выбран язык программирования C++, т. к. благодаря своей вычислительной мощности язык обеспечивает высокую скорость исполнения кода, не утяжеляет программы и позволяет использовать их даже на старых устройствах.

2.5. Определение необходимости проведения научно-исследовательских работ на последующих стадиях.

– Не требуется.

2.6. Согласование и утверждение технического задания

Исполнитель _____

Заказчик _____

7.9.2. Спецификация *(недостаток – не соблюдена рубрикация).*

Оглавление

1. Постановка задачи

Написать программы на языке C++ (особенности в *Техническом Задании*), которые необходимы для решения следующих шести задач (спецификация головной программы: файл) *гиперссылка).*

1.1. Первая задача

Задача 18. Написать программу для вычисления кинетической энергии E_k релятивистского электрона. Использовать формулу

$$E_k = \frac{m_e c^2}{\sqrt{1 - \left(v/c\right)^2}} - m_e c^2,$$

где масса (масса покоя) электрона $m_e \approx 9.1 \times 10^{-31}$ (кг), скорость света $c \approx 2.998 \times 10^8$ (м/с), а v – скорость электрона. Для сравнения рассчитайте нерелятивистское выражение $E = \frac{m_e v^2}{2}$.

 Описание программы:

- Программа должна вычислять кинетическую энергию релятивистского электрона и его кинетическую энергию при нерелятивистском движении.
- Для расчета релятивистской энергии используется формула: Ek = (m * pow(c, 2)) / (sqrt(1 - pow((v / c), 2))) - m * pow(c, 2).
- Для нерелятивистской кинетической энергии: E = (m * pow(v, 2)) / 2.
- Где m – масса электрона, c – скорость света, v – скорость электрона.

 Шаги для реализации:

1. **Определение констант:**
   - double m = 9.1 * pow(10, -31); // масса электрона в кг;
   - double c = 2.998 * pow(10, 8); // скорость света в м/с.
2. **Ввод данных:**
   - Создать переменную v типа double для хранения скорости электрона.
   - Ввести скорость электрона либо через консоль, либо считать её из файла:
     - Если данные вводятся вручную, использовать функцию для проверки корректности данных (например, скорость должна быть в пределах от 0 до 10000 м/с):
       - double v;
       - cout << "Введите скорость v электрона (м/с): ";
       - CorrectInput(v, 1);
     - Если данные берутся из файла, использовать функцию для чтения данных из файла:
       - ifstream fin("data1.txt");
       - ReadDataFromFile(fin, "v", v, "Данные взяты из файла data1.txt.", 1);
3. **Рассчитать релятивистскую кинетическую энергию:**
   - Использовать переменные для хранения результата:
     - Ek типа double для хранения релятивистской энергии.
   - Вычислить по формуле: Ek = (m * pow(c, 2)) / (sqrt(1 - pow((v / c), 2))) - m * pow(c, 2);
   - Важно добавить проверку, чтобы v не превышала c, иначе произойдет ошибка при вычислении.
4. **Рассчитать нерелятивистскую кинетическую энергию:**
   - Создать переменную E типа double для хранения нерелятивистской энергии.
   - Вычислить по формуле: E = (m * pow(v, 2)) / 2;
5. **Вывод результатов:**
   - Вывести рассчитанные значения релятивистской и нерелятивистской кинетической энергии на экран.
   - Записать результаты вычислений в файл output1.txt:
     - ostringstream file_output;
     - file_output << "Кинетическая энергия Ek релятивистского электрона = " << Ek << " Дж\n";
     - file_output << "Нерелятивистская энергия E = " << E << " Дж\n";
     - WriteToFile("output1.txt", file_output).

1.2. Вторая задача

Задача 16. Написать программу для вычисления методом последовательных итераций уравнения $x = A\exp(-x)$. Параметр A вводится пользователем. Проверить, для каких значений параметра A применим метод последовательных итераций.

Метод последовательных итераций должен использоваться для приближенного решения уравнений. В данном случае рассматривается уравнение $x=A\cdot\exp(-x)$:

1. **Выбор начального приближения**:
   - Начальное значение переменной x выбрать исходя из значения параметра A.
     - Если A < e, начальное значение x = 0.
     - Если A ≥ e, начальное значение x = 1.
2. **Итерационная формула**:
   - После выбора начального значения итерации будут проводиться с использованием рекуррентной формулы:
     - Если A < e, используйте формулу $x_{i+1}=A/\exp(x_i)$.
     - Если A ≥ e, используйте формулу $x_{i+1}=\log(A)-\log(x_i)$.
   - На каждом шаге будет вычисляться новое значение x, которое должно постепенно приближаться к решению уравнения.
3. **Остановка итераций**:

Итерации должны повторяться до тех пор, пока значение x не стабилизируется, то есть пока разница между последовательными значениями x не станет достаточно малой, или до достижения максимального числа итераций.

1.3. Третья задача

Задача 9. Моментом порядка k случайной величины ξ называется число $M\xi^k = \sum_{i=1}^{n} \xi_i^k p_i$, где случайная величина ξ принимает значения ξ_1, ξ_2, ..., ξ_n с вероятностями p_1, p_2, ..., p_n соответственно. Написать программу для вычисления момента порядка k для случайной величины с заданным законом распределения. Порядок момента и закон распределения вводятся пользователем.

Алгоритм выполнения задачи:

1. Пользователь должен ввести количество возможных значений случайной величины nnn, порядок момента k, а также значения ξi и pi, либо данные можно считать из файла.
2. После ввода данных мы рассчитываем момент порядка k с помощью формулы, данной в задании.
3. Программа должна поддерживать ввод данных как с клавиатуры, так и из файла.

Определение переменных:

- Создать переменную n типа double – количество возможных значений случайной величины (максимум 100).
- Создать переменную k типа double – порядок момента.
- Создать массивы xi[101] и p[101] типа double – для хранения значений случайной величины и их вероятностей (размер 101, так как максимальное значение nnn – 100).
- Создать переменную Mxik типа double для хранения результата вычисления момента (инициализировать значением 0).

Данные вводятся с консоли или из файла:

- Ввод с консоли:
  - Получить число реализуемых значений случайной величины n и k при помощи функции CorrectInput.
  - Ввести значения случайной величины xi и их вероятности p при помощи цикла for (int i = 0; i < n; i++) и функции CorrectInput.

- Ввод с файла:
  - ifstream fin("data3.txt");
  - ReadDataFromFile(fin, "n", n, "", 1);
  - ReadDataFromFile(fin, "k", k, "Значения xi:", 1);
  - ReadDataFromFile(fin, "xi", xi, n, "Вероятности p:", 1);
  - fin.close();

Вычисление момента:

for (int i = 0; i < n; i++) Mxik += p[i] * pow(xi[i], k); // вычисление момента порядка k

Вывод на экран (cout) и в файл:

- ostringstream file_output;
- file_output << "\nМомент (Mxik) порядка " << k << " = " << Mxik << endl;
- WriteToFile("output3.txt", file_output);

1.4. Четвертая задача

Задача 12. Написать программу с функцией для поиска экстремального (наибольшего или наименьшего) элемента массива. Массив заполнить случайными числами.

 Начальная инициализация экстремальных значений:

- Инициализируйте переменные для максимального и минимального элементов, присваивая им значение первого элемента массива (например, max = arr[0] и min = arr[0]).

 Поиск экстремальных значений:

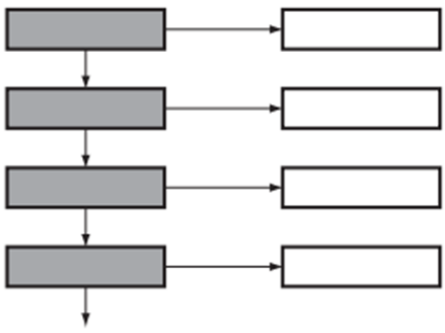
- Проходим по массиву в цикле от второго элемента до последнего (с 1 до size-1).
- На каждом шаге сравниваем текущий элемент с текущим максимумом и минимумом:
  - Если текущий элемент больше текущего максимума, обновляем значение максимума.
  - Если текущий элемент меньше текущего минимума, обновляем значение минимума.

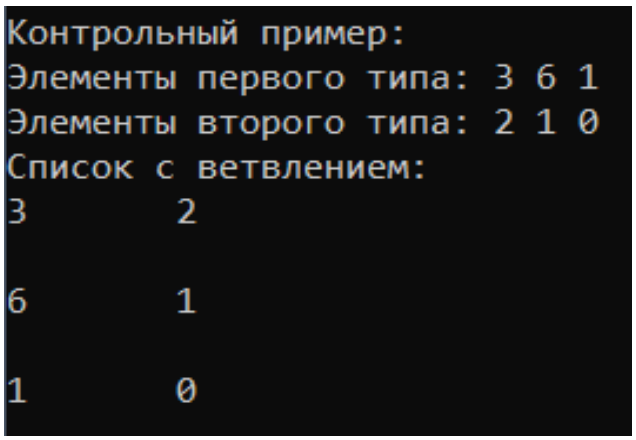
1.5. Пятая задача

Задача 12. Написать программу для транспонирования матрицы, реализованной в виде двумерного динамического массива.

1.6. Шестая задача

Написать программу для создания динамической конструкции из элементов двух типов (Список с ветвлением). Элементы первого типа ссылаются на два элемента. Элементы второго типа не содержат полей для выполнения ссылок. Элементы первого типа образуют последовательный список. Каждый из элементов этого списка ссылается на следующий элемент и на элемент второго типа.





1. **Определение структур**: Начните с определения двух связанных структур:
   - **Type2Node**: одиночный узел для второго типа, который содержит данные типа **double**.
   - **Type1Node**: узел для первого типа, который содержит данные, указатель на следующий узел этого же типа и указатель на узел второго типа.
2. **Создание списка**:
   - Определите указатель на голову списка (**head**) как **nullptr**. Также определите указатель для текущего узла (**current**), который поможет добавлять новые узлы в список.
3. **Ввод данных**:
   - Реализуйте механизм для ввода данных, который позволит создавать динамические узлы. Если данные берутся из консоли, циклом запрашивайте количество пар данных от пользователя.
   - Для каждой итерации создавайте новый узел **Type1Node** и новый узел **Type2Node**:
     - Запрашивайте значения для обоих узлов и сохраняйте их.
     - Устанавливайте ссылку (**ref**) в узле первого типа на только что созданный узел второго типа.
4. **Связывание узлов**:
   - Если указатель **head** равен **nullptr**, присвойте его новому узлу первого типа. Если нет, связывайте узлы: устанавливайте **next** предыдущего узла на новый узел и обновляйте указатель **current**.
5. **Вывод списка**:
   - После завершения ввода данных реализуйте вывод элементов списка, который чередует элементы первого и второго типов.
   - Определите очередной вывод в файл.
6. **Освобождение памяти**:
   - Обязательно включите код для освобождения памяти. Перебирайте список узлов, удаляя каждый узел второго типа перед удалением узла первого типа.

2. Входные и выходные данные для задач

Основное меню:

Переменные **Dchoice** типа double (Используется для проверки корректного ввода, так как функция корректного ввода работает с числами типа double) и **choice** типа int (этой переменной присваивается значение Dchoice, предварительно проверенное функцией защиты от некорректного ввода как целое число, удовлетворяющее нужному диапазону).

Файлы с данными для ввода и с выходными данными представлены в папке «ФИО лаба1» (пронумерованы соответственно номеру задачи, т. е. data1.txt, output3.txt и т. д.)

***Для чтения данных с файла использовать функцию ReadDataFromFile, для записи данных в файл – функция WriteToFile. Для защиты от некорректного ввода использовать функцию CorreceInput. Принцип их работы подробно разобран в спецификации дополнительных функций.**

2.1. Первая задача

Входные данные:

Файл, содержащий данные для чтения: data1.txt

Переменные типа double:

- m (начальное значение 9.1 * 10^(-31), масса электрона в кг);
- c (начальное значение 2.998 * 10^8, скорость света в м/с);
- v (скорость электрона, вводится пользователем);
- otkyda (переменная для выбора источника данных: консоль или файл).

Промежуточные данные:

- Ek – переменная типа double (переменная для хранения релятивистской кинетической энергии, вычисляемая по формуле Ek = (m * pow(c, 2)) / (sqrt(1 - pow((v / c), 2))) - m * pow(c, 2);).
- E – переменная типа double (переменная для хранения нерелятивистской кинетической энергии, вычисляемая по формуле E = (m * pow(v, 2)) / 2;).
- file_output: строковый поток, используемый для формирования строки для записи в файл.

Выходные данные:

- Переменные типа double:
  - Ek, предназначенная для вывода результата программы – релятивистская кинетическая энергия в джоулях.
  - E, предназначенная для вывода результата программы – нерелятивистская кинетическая энергия в джоулях.
- Запись результатов в файл "output1.txt" в формате:
  - Программа для вычисления кинетической энергии релятивистского электрона.
  - Кинетическая энергия Ek релятивистского электрона = [значение] Дж.
  - Нерелетявистское движение E = [значение] Дж.

Файл, в который выводится результат output1.txt

Программа для вычисления кинетической энергии релятивистского электрона по формуле: Ek = (m*c^2) / (sqrt(1-(v/c)^2)) - m*c^2.

Также в программе вычисляется нерелятивистское движение по формуле: E = (m*v^2) / 2.

m = 9.1 * 10^(-31) кг – масса электрона; c = 2.998 * 10^8 м/с – скорость света.

Кинетическая энергия Ek релятивистского электрона = 8.58281e-28 Дж.

Нерелятивистское движение E = 8.60974e-28 Дж.

2.2. Вторая задача

Входные данные:

Файл, содержащий данные для чтения data2.txt

- A: переменная типа double, параметр для уравнения, вводится пользователем или читается из файла.
- otkyda: переменная типа double, определяющая источник данных (1 – консоль, 2 – файл).
- N: переменная типа int, фиксированное количество итераций (100).
- precision: переменная типа double, задающая точность вычислений (10^-6).
- x: переменная типа double, текущее значение (инициализируется в зависимости от значения A).
- prev_x: переменная типа double, предыдущее значение x.

Промежуточные данные:

- prev_x: используется для хранения предыдущего значения x в цикле.
- x: обновляется в цикле по формуле:
  - Для A < e:
    - Инициализация x=0.
    - Цикл for (с i=1 до N):

- Вычисление нового значения x по формуле x = A / pow(e, x); и проверка на точность при помощи if (fabs(x - prev_x) < precision).
  - Для A ≥ e:
    - Инициализация x=0.
    - Цикл for (с i=1 до N):
- Вычисление нового значения x по формуле x = log(A) - log(x); и проверка на точность при помощи if (fabs(x - prev_x) < precision).
- i: переменная типа int, счётчик для цикла (for) итераций (1 до N включительно). Начальное значение: 1.
- Цикл for (i = 1; i <= N; i++): используется для выполнения 100 итераций вычисления корня уравнения.
- file_output: строковый поток, используемый для формирования строки для записи в файл.

Выходные данные:

 Вывод в консоль: Значение корня уравнения **x** после 100 итераций или до достижения точности.

- Запись результатов в файл output2.txt в формате:
  - Программа для вычисления методом последовательных итераций.
  - A = [значение].
  - x = [значение].

Файл, в который выводится результат output2.txt

Программа для вычисления методом последовательных итераций уравнения x = A * exp(-x).

Задается 100 итераций; точность: 10^(-6)

A = 22.5

x = 2.2865

2.3. Третья задача

Входные данные:

Файл, содержащий данные для чтения data3.txt

- n: переменная типа double, число реализуемых значений случайной величины (0 < n <= 100), вводится пользователем или читается из файла.
- k: переменная типа double, порядок момента (0 < k <= 100), вводится пользователем или читается из файла.
- otkyda: переменная типа double, определяющая источник данных (1 – консоль, 2 – файл).
- xi[101]: массив типа double для хранения значений случайной величины.
- p[101]: массив типа double для хранения вероятностей соответствующих значений.
- Mxik: переменная типа double (начальное значение 0).

Промежуточные данные:

- Mxik: переменная типа double, вычисляется по формуле: Mxik += p[i] * pow(xi[i], k).
- file_output: строковый поток, используемый для формирования строки для записи в файл.
- i: переменная типа int, счётчик для цикла, используемый для вычисления момента порядка k.
- Цикл for (i = 0; i < n; i++): используется для ввода значений xi и p, а также для вычисления момента Mxik.
- file_output: строковый поток для записи в файл.

Выходные данные:

- Вывод в консоль:
  - Mxik: вычисленный момент (в зависимости от введённых значений и вероятностей).
- Запись в файл "output3.txt" в формате:
  - Программа для вычисления момента порядка k.
  - n = [значение]
  - k = [значение]

- xi_1 = [значение]; p_1 = [значение];
- ...
- Момент (Mxik) порядка k = [значение]

Файл, в который выводится результат output3.txt

Программа для вычисления момента порядка k для случайной величины с заданным законом распределения:

Mxik = (xi_1)^k * p_1 + (xi_2)^k * p+2 + ... + (xi_n)^k * p_n

К примеру, для трёх значений случайных величин 1, 3, 5 с вероятностями 0.3, 0.2, 0.5 соответственно, момент порядка 4 будет равен: 1^4*0.3 + 3^4*0.2 + 5^4*0.5 = 329.

n = 3

k = 4

xi_1 = 1; p_1 = 0.3;

xi_2 = 3; p_2 = 0.2;

xi_3 = 5; p_3 = 0.5;

Момент (Mxik) порядка 4 = 329.

2.4. Четвертая задача

Входные данные:

Файл, содержащий данные для чтения data4.txt

- N: константа, задающая максимальный размер массива (10000).
- arr[N]: массив типа double, используемый для хранения случайных целых чисел от 0 до 999.
- size: переменная типа double, размер массива, вводится пользователем.
- otkyda: переменная типа double, определяющая источник данных (1 – консоль, 2 – файл).

Промежуточные данные:

- max: переменная типа int, используется для хранения максимального элемента массива.
- min: переменная типа int, используется для хранения минимального элемента массива.
- Цикл for (int j = 0; j < size; j++):
  - j: целочисленный индекс массива, используется для заполнения массива случайными числами или для чтения данных из файла.
- Цикл for (int j = 1; j < size; j++):
  - j: целочисленный индекс массива, используется для поиска максимального и минимального элементов массива.
- file_output: строковый поток для записи в файл.

Выходные данные:

- Вывод в консоль:
- Выводится сообщение с максимальным элементом массива и минимальным элементом массива в формате:
  - Максимальный элемент массива = <значение>
  - Минимальный элемент массива = <значение>
- Запись в файл "output4.txt": создается строка с описанием программы и массивом. Содержимое файла включает:
  - Программа для поиска экстремального элемента массива.
  - Массив: <значения элементов массива>
  - Максимальный элемент массива = <значение>
  - Минимальный элемент массива = <значение>.

Файл, в который выводится результат output4.txt

Программа для поиска экстремального элемента массива.

Массив: 30 10 2134 3 432 52 1234 213 2 102 8 3 34 4 1 2 3 321837 323 3829 23 21 22 3 0

Максимальный элемент массива = 321837

Минимальный элемент массива = 0

2.5. Пятая задача

Входные данные:

Файл, содержащий данные для чтения data5.txt

- n: переменная типа double, количество строк матрицы, вводится пользователем или читается из файла.
- m: переменная типа double, количество столбцов матрицы, вводится пользователем или читается из файла.
- otkyda: переменная типа double, определяющая источник данных (1 – консоль, 2 – файл).
- nm: динамический массив указателей типа double, используется для хранения элементов матрицы.

Промежуточные данные:

- file_output: строковый поток для записи в файл.
- Цикл for (int i = 0; i < n; i++):
  - i: целочисленный индекс строки, используется для инициализации строк матрицы.
- Цикл for (int i = 0; i < n; i++) (ввод элементов):
  - i: целочисленный индекс строки, используется для ввода элементов матрицы с консоли.
- Цикл for (int j = 0; j < m; j++) (ввод элементов):
  - j: целочисленный индекс столбца, используется для ввода элементов матрицы с консоли.
- Цикл for (int i = 0; i < n; i++) (вывод исходной матрицы):
  - i: целочисленный индекс строки, используется для вывода элементов исходной матрицы.
- Цикл for (int i = 0; i < m; i++) (вывод транспонированной матрицы):
  - i: целочисленный индекс столбца, используется для вывода элементов транспонированной матрицы.
- Цикл for (int j = 0; j < n; j++) (вывод транспонированной матрицы):
  - j: целочисленный индекс строки, используется для вывода элементов транспонированной матрицы.

Выходные данные:

- Вывод в консоль:
  - Исходная матрица выводится в формате:
    - Исходная матрица:
    - <значения элементов матрицы>
  - Транспонированная матрица выводится в формате:
    - Транспонированная матрица:
    - <значения элементов транспонированной матрицы>
- Запись в файл "output5.txt":
  - Содержимое файла включает:
    - Программа для транспонирования матрицы.
    - Исходная матрица:
    - <значения элементов матрицы>
    - Транспонированная матрица:
    - <значения элементов транспонированной матрицы>

Файл, в который выводится результат output5.txt

Программа для транспонирования матрицы.

Исходная матрица:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
1	1	1	1	1
0	0	0	0	0

Транспонированная матрица:

1	6	11	16	1	0
2	7	12	17	1	0
3	8	13	18	1	0
4	9	14	19	1	0
5	10	15	20	1	0

2.6. Шестая задача

Входные данные:

Файл, содержащий данные для чтения data6.txt

- Type1Node и Type2Node: структуры для создания узлов списка.
- head: указатель на голову списка типа Type1Node.
- current: указатель для перемещения по списку.
- n: переменная типа double, количество пар элементов, вводится пользователем или читается из файла.
- otkyda: переменная типа double, определяющая источник данных (1 – консоль, 2 – файл).

Промежуточные данные:

- file_output: строковый поток для записи в файл.
- Цикл for (int i = 0; i < n; i++):
  - i: целочисленный индекс, используется для создания узлов списка и ввода данных.
- Цикл while (current != nullptr):
  - current: целочисленный указатель на текущий элемент списка, используется для вывода списка с ветвлением.

Выходные данные:

- Вывод в консоль:
  - Элементы списка с ветвлением выводятся в формате:
    - Список с ветвлением:
    - <значения первого типа> <значения второго типа>
- Запись в файл "output6.txt":
  - Содержимое файла включает:
    - Программа для создания динамической конструкции из элементов двух типов (Список с ветвлением).
    - Список с ветвлением:
    - <значения первого типа> <значения второго типа>

Файл, в который выводится результат output6.txt

Программа для создания динамической конструкции из элементов двух типов (Список с ветвлением).

Список с ветвлением:

1	6
2	4
3	1
9	0

3. Корректные и ошибочные данные

- Корректные данные **(общие для всех подпрограмм)**
  - Выбор способа ввода данных (otkyda): используем подпрограмму CorrectInput(otkyda, 2):
    - 1 – ввод данных с клавиатуры
    - 2 – чтение данных из файла
- Ошибочные данные **(общие для всех подпрограмм)**
  - Выбор способа ввода данных:
    - 0 (не соответствует 1 или 2)

- Нечисловой ввод везде, где осуществляется пользовательский ввод:
  - "abc" (ввод строки вместо числа)
  - "10a" (ввод числа с буквой)
  - "3.14abc" (дробное число с буквой)
  - "" (пустая строка)

Первая подпрограмма: «Кинетическая энергия релятивистского электрона»

- **Используем подпрограмму**: CorrectInput(v, 1);
- **Проверка**: 0 < v <= 10000

Корректные данные:

- v = 1
- v = 5000
- v = 10000

Ошибочные данные:

- v = 0 (не больше 0)
- v = 10001 (больше 10000)
- v = -1 (отрицательное значение)
- v = 15000.5 (больше 10000, дробное число)

Вторая подпрограмма: «Метод последовательных итераций»

Используем подпрограмму: CorrectInput(A, 1);
Проверка: 0 < A <= 10000
Корректные данные:
A = 1
A = 100
A = 9999.99
Ошибочные данные:
A = 0 (не больше 0)
A = 10001 (больше 10000)
A = -5 (отрицательное значение)
A = 5000.5 (дробное значение, но в пределах)

Третья подпрограмма: «Момент случайной величины»

Используем подпрограмму: CorrectInput(n, 3); и CorrectInput(k, 3);
Проверка для n: 0 < n <= 100
Проверка для k: 0 < k <= 100
Проверка для xi[i]: 0 < xi[i] <= 10000
Проверка для p[i]: 0 <= p[i] <= 1
Корректные данные:
n = 3, k = 4
xi[i]: 1 3 5
p[i]: 0.3 0.2 0.5
n = 2, k = 10
xi[i]: 7 61
p[i]: 0.8 0.2
Ошибочные данные:
Ошибки для n:
n = 0 (не больше 0)
n = 101 (больше 100)
Ошибки для k:
k = 0 (не больше 0)
k = 101 (больше 100)
Ошибки для xi[i]:
xi[i] = 0 (не больше 0)
xi[i] = 10001 (больше 10000)

Ошибки для p[i]:
xi[i] = -1 (не 0 или больше)
xi[i] = 1.001 больше 1)

Четвертая подпрограмма: «Поиск максимального и минимального элемента массива»

- **Используем подпрограмму**: CorrectInput(size, 11);
- **Проверка**: 0 < size <= 10000
- Корректные данные:
  - size = 1
  - size = 500
  - size = 10000

Ошибочные данные:

- size = 0 (не больше 0)
- size = 10001 (больше 10000)
- size = -1 (отрицательное значение)
- size = 0.5 (дробное значение)

Пятая подпрограмма: «Транспонирование матрицы»

Используем подпрограмму: CorrectInput(n, 5);, CorrectInput(m, 5) для размеров матрицы;
Проверка для n и m: $0 < n, m \leq 40$
Корректные данные:
n = 1, m = 1
n = 10, m = 20
n = 40, m = 40
Ошибочные данные:
n = 0 (не больше 0)
m = 41 (больше 40)
n = -1 (отрицательное значение)
m = 15.5 (дробное значение)
Используем подпрограмму для элементов матрицы: CorrectInput(nm[i][j], 1);
Проверка для элементов матрицы: $1 \leq nm[i][j] \leq 10000$
Корректные данные для элементов матрицы:
nm[1][1] = 1
nm[5][5] = 500
nm[10][10] = 9999
Ошибочные данные для элементов матрицы:
nm[2][3] = 0 (не больше 0)
nm[3][4] = 10001 (больше 10000)
nm[5][5] = -5 (отрицательное значение)
nm[4][6] = 5000.5 (дробное значение)

Шестая подпрограмма: «Список с ветвлением»

- Используем подпрограмму: CorrectInput(n, 5); для ввода количества пар элементов.
- Используем подпрограмму: CorrectInput(newNode->data, 1); для ввода данных первого типа (Type1Node).
- Используем подпрограмму: CorrectInput(newRefNode->data, 1); для ввода данных второго типа (Type2Node).
- Проверка для количества пар элементов (n):
  - $0 < n \leq 40$
- Корректные данные для количества пар элементов (n):
  - n = 1
  - n = 10
  - n = 40
- Ошибочные данные для количества пар элементов (n):
  - n = 0 (не больше 0)

- n = 41 (больше 40)
- n = -3 (отрицательное значение)
- n = 3.5 (дробное значение)

- Проверка для элементов первого типа (Type1Node):
  - $1 \leq data \leq 10000$
- Корректные данные для элементов первого типа (Type1Node):
  - data = 1
  - data = 5000
  - data = 10000
- Ошибочные данные для элементов первого типа (Type1Node):
  - data = 0 (не больше 0)
  - data = 10001 (больше 10000)
  - data = -10 (отрицательное значение)
  - data = 6000.5 (дробное значение)
- Проверка для элементов второго типа (Type2Node):
  - $1 \leq data \leq 10000$
- Корректные данные для элементов второго типа (Type2Node):
  - data = 1
  - data = 2000
  - data = 9999
- Ошибочные данные для элементов второго типа (Type2Node):
  - data = 0 (не больше 0)
  - data = 10001 (больше 10000)
  - data = -7 (отрицательное значение)
  - data = 3000.75 (дробное значение)

В случае некорректного ввода, пользователь должен увидеть сообщение: «Некорректный ввод. Пожалуйста, попробуйте ввести данные заново».

4. Пользователь программы

Пользователем программ могут являться представители заказчика, ученики старших классов, студенты и их преподаватели.

5. Требования к интерфейсу

5.1. Язык и вид интерфейса

Язык, используемый для общения с пользователем – русский, согласно техническому заданию. Интерфейс понятен для пользователя. Он представляет собой текст интерфейса с белым шрифтом букв на черном фоне, чтобы быть читабельным и практичным.

5.2. Описание интерфейса

*При вводе некорректных значений (описаны в п. 3), пользователь получит сообщение «Некорректный ввод. Пожалуйста, попробуйте ввести данные заново».

Главное меню программы должно быть следующим:

- "В программе представлена возможность для решения 7 различных задач, представленных в меню.\n";
- "Программы 1-6 и головную программу написал студент ФИО (ИБ_2).\n";
- "Программу 7 написал другой студент ФИО (ИБ_2)\n";
- "\nМеню:\n";
- "1. Кинетическая энергия релятивистского электрона\n";
- "2. Решение уравнения x = A * exp(-x) методом последовательных итераций\n";
- "3. Момент порядка k\n";
- "4. Поиск экстремального элемента массива\n";
- "5. Транспонирование матрицы\n";
- "6. Список с ветвлением\n";
- "7. Нахождение числа Фибоначчи по его номеру\n";

- "0. Выход\n";
- "Выберите номер программы (1-7): ";

5.2.1. Если пользователь введет 1, то программа должна выводить на экран:

- Программа для вычисления кинетической энергии релятивистского электрона по формуле:
- Ek = (m*c^2) / (sqrt(1-(v/c)^2)) - m*c^2.
- Также в программе вычисляется нерелятивистское движение по формуле: E = (m*v^2) / 2
- m = 9.1 * 10^(-31) кг – масса электрона; c = 2.998 * 10^8 м/с – скорость света
- Откуда брать данные? (1 – консоль, 2 – файл):

Если пользователь выбрал цифру 1, то вывод текста:

- Введите скорость v электрона в м/с (0 < v <= 10000):
- Кинетическая энергия Ek релятивистского электрона = [Ek] Дж
- Нерелетявистское движение E = [E] Дж
- Данные записаны в файл output1.txt в папке с программой.

Если пользователь выбрал цифру 2, то вывод текста:

- v =
- Данные взяты из файла data1.txt в папке с программой.
- Кинетическая энергия Ek релятивистского электрона = [Ek] Дж
- Нерелетявистское движение E = [E] Дж
- Данные записаны в файл output1.txt в папке с программой.

5.2.2. Если пользователь введет 2, то программа должна выводить:

- Программа для вычисления методом последовательных итераций уравнения x = A * exp(-x).
- Задается 100 итераций; точность: 10^(-6)
- Откуда брать данные? (1 - консоль, 2 - файл):

Если пользователь выбрал цифру 1, то вывод текста:

- Введите значение параметра A (0 < A <= 10000):
- x =
- Данные записаны в файл output2.txt в папке с программой.

Если пользователь выбрал цифру 2, то вывод текста:

- A =
- Данные (A) взяты из файла data2.txt в папке с программой.
- x =
- Данные записаны в файл output2.txt в папке с программой.

5.2.3. Если пользователь введет 3, то программа должна выводить:

- Программа для вычисления момента порядка k для случайной величины с заданным законом распределения:
- Mxik = (xi_1)^k * p_1 + (xi_2)^k * p+2 + ... + (xi_n)^k * p_n
- К примеру, для трёх значений случайных величин 1, 3, 5 с вероятностями 0.3, 0.2, 0.5 соответственно, момент порядка 4 будет равен: 1^4*0.3 + 3^4*0.2 + 5^4*0.5 = 329.
- Откуда брать данные? (1 – консоль, 2 – файл):

Если пользователь выбрал цифру 1, то вывод текста:

- Введите число реализуемых значений случайной величины (целое число n, 0 < n <= 100):
- Введите порядок момента (целое число k, 0 < k <= 100):
- Введите возможные значения случайной величины (xi, 0 < xi <= 10000):
- xi_1 =

... // нужное количество значений xi

- Введите вероятности (p, 0 <= p <= 1):
- p_1 =

... // нужное количество значений p

- Момент (Mxik) порядка 3 = 29.4
- Данные записаны в файл output3.txt в папке с программой.

Если пользователь выбрал цифру 2, то вывод текста:

- n =
- k =
- Значения случайной величины xi:
- xi:
- Значения вероятностей p:
- p:
- Данные взяты из файла data3.txt в папке с программой.
- Момент (Mxik) порядка [k] =
- Данные записаны в файл output3.txt в папке с программой.

5.2.4. Если пользователь введет 4, то программа должна выводить:

- Программа для поиска экстремального элемента массива.
- Массив заполняется случайными целыми числами от 0 до 999.
- Откуда брать данные? (1 – консоль, 2 – файл):

Если пользователь выбрал цифру 1, то вывод текста:

- Введите размер массива (от 1 до 10000):
- Массив:
- Максимальный элемент массива =
- Минимальный элемент массива =
- Данные записаны в файл output4.txt в папке с программой.

Если пользователь выбрал цифру 2, то вывод текста:

- size =
- arr:
- Данные взяты из файла data4.txt в папке с программой.
- Максимальный элемент массива =
- Минимальный элемент массива =
- Данные записаны в файл output4.txt в папке с программой.

5.2.5. Если пользователь введет 5, то программа должна выводить:

- Программа для транспонирования матрицы.
- Откуда брать данные? (1 – консоль, 2 – файл):

Если пользователь выбрал цифру 1, то вывод текста:

- Введите количество строк (n, 0 < n <= 40) матрицы:
- Введите количество столбцов (m, 0 < n <= 40) матрицы:
- Введите элементы матрицы (числа от 1 до 10000):
- Исходная матрица:
- Транспонированная матрица:
- Данные записаны в файл output5.txt в папке с программой.

Если пользователь выбрал цифру 2, то вывод текста:

- Данные взяты из файла data5.txt в папке с программой.
- n =
- m =
- Исходная матрица:
- Транспонированная матрица:
- Данные записаны в файл output5.txt в папке с программой.

5.2.6. Если пользователь введет 6, то программа должна выводить:

- Программа для создания динамической конструкции из элементов двух типов (Список с ветвлением).

- Контрольный пример:
- Элементы первого типа: 3 6 1
- Элементы второго типа: 2 1 0
- Список с ветвлением:
- 3 2
- 6 1
- 1 0
- Откуда брать данные? (1 – консоль, 2 – файл):

Если пользователь выбрал цифру 1, то вывод текста:

- Введите количество пар элементов в списке (целое число n, 0 < n <= 40):
- Введите 1-й элемент ПЕРВОГО типа (число от 1 до 10000):

Введите 1-й элемент ВТОРОГО типа (число от 1 до 10000):

- Введите 2-й элемент ПЕРВОГО типа (число от 1 до 10000):

Введите 2-й элемент ВТОРОГО типа (число от 1 до 10000):

… // нужное количество элементов

- Список с ветвлением:
- Данные записаны в файл output6.txt в папке с программой.

Если пользователь выбрал цифру 2, то вывод текста:

- Количество пар элементов списка =
- Элементы списка (чередуются - сначала первого типа, затем - второго):
- Данные взяты из файла data6.txt в папке с программой.
- Список с ветвлением:
- Данные записаны в файл output6.txt в папке с программой.

6. Необходимая документация

Отчет должен включать в себя:

• Текст готовой программы в отдельном файле «код» на языке C++, который был выбран в Техническом задании.

• Скриншоты программы (файл в формате Microsoft Word Лабораторная 1).

• Вывод о соответствии предоставленной программы ТЗ и спецификации, Степень соответствия реализации технологического процесса создания программного продукта в коллективе.

7. Перспективы развития программы

Программа может быть доработана по требованию заказчика. Оплата в двойном размере.

Спецификация головной программы *Рубрикация как таковая отсутствует, однако текст можно было включить в основную спецификацию как подраздел.*

(int main()):

Головная программа: Меню с выбором задачи

Входные данные:

– Dchoice: переменная типа double, используется для выбора номера программы. Вводится пользователем.

– choice: переменная типа int, хранит преобразованное значение Dchoice и используется для обработки выбора пользователя.

Промежуточные данные:

– Цикл do-while: управляет процессом выбора программы. Позволяет пользователю повторно выбирать задачи до тех пор, пока не будет введён 0 для выхода.

– Переменная choice: принимает целочисленное значение от Dchoice, которое сравнивается в конструкции switch для выполнения выбранной программы.

Обработка ошибок:

– Если ввод не соответствует допустимым значениям (0–7), программа выводит сообщение: «Некорректный выбор. Попробуйте снова».

– Используется подпрограмма CorrectInput(Dchoice, 0) для проверки корректности ввода. Она обеспечивает правильный тип и диапазон введённых данных.

Меню задач:

1. Кинетическая энергия релятивистского электрона – функция KineticEnergy().
2. Решение уравнения методом последовательных итераций – функция IterationMethod().
3. Момент порядка k – функция MomentCalculation().
4. Поиск экстремального элемента массива – функция ExtremElementSearch().
5. Транспонирование матрицы – функция MatrixTranspose().
6. Список с ветвлением – функция BranchingList().
7. Нахождение числа Фибоначчи по его номеру – функция FibonacciProgram().

Выходные данные:

– Вывод соответствующего сообщения в консоль при каждом выполнении программы.

– Программа завершает выполнение, если пользователь выбирает 0, при этом выводится сообщение: «Программа завершена. Спасибо за использование =)».

Ошибочные данные:

– Ввод значений вне диапазона (не от 0 до 7).

– Неверный тип данных (не числовое значение), обрабатываемый через подпрограмму CorrectInput.

Корректные данные:

– Выбор номера программы в пределах от 0 до 7.

– Программа завершается корректно при выборе 0.

7.9.3. Спецификация дополнительных функций

1.2.3. Функция CorrectInput(double& p1, int userflag)

Проверяет ввод пользователя и обеспечивает защиту от некорректного ввода. Описание:

Название: CorrectInput

Аргументы:

double& p1: ссылка на переменную типа double, в которую будет записано корректное значение.

int userflag: флаг, определяющий условия проверки входных данных.

Логика работы функции:

Функция должна бесконечно запрашивать ввод у пользователя до тех пор, пока не будет введено корректное значение.

Создается строковая переменная input, в которую будет считываться пользовательский ввод.

Функция входит в бесконечный цикл while (true), который будет продолжаться до тех пор, пока не произойдет корректный ввод.

С помощью getline считывается строка, введенная пользователем.

В цикле по каждому символу строки проверяется, есть ли запятые, и если такие встречаются, они заменяются на точки (для корректного восприятия чисел с плавающей запятой):

```
for (size_t i = 0; i < input.size(); ++i)
    if (input[i] == ',') input[i] = '.';
```

Создается поток stringstream ss(input), с помощью которого строка будет преобразована в числовое значение.

Объявляются переменные value для хранения числового значения и extra для проверки на наличие лишних символов в строке.

В зависимости от значения userflag, необходимо реализовать различные условия проверки:

userflag == 1: значение должно находиться в диапазоне [1, 10000].

userflag == 11: значение должно находиться в диапазоне [1, 10000] и быть целым числом.

userflag == 2: значение должно быть либо 1, либо 2.

userflag == 3: значение должно быть в диапазоне (0, 100].

userflag == 4: значение должно быть в диапазоне [0, 1].

userflag == 5: значение должно быть в диапазоне [1, 40].

В случае значений userflag 3, 5 или 0, значение должно быть целым числом (должно быть равно своему целочисленному представлению).

Если хотя бы одно условие не выполнено, выводится сообщение о некорректном вводе, и цикл продолжается.

Если ввод корректен, значение записывается в переменную p1, и цикл завершается.

В случае некорректного ввода (например, если в строке содержатся не только числа или имеются лишние символы), выводится сообщение об ошибке ("Некорректный ввод. Пожалуйста, попробуйте ввести данные заново."), и пользователю предлагается повторить ввод.

После каждого некорректного ввода поток cin очищается и сбрасывается для устранения ошибок ввода.

Пример вызова функции:

```
double number;
// Допустим, нужен ввод числа от 1 до 40
CorrectInput(number, 5);
```

1.2.4. Функция ReadDataFromFile (перегруженная) для чтения переменной и массива из файла

1. Перегруженная функция для чтения переменной из открытого потока:

   void ReadDataFromFile(ifstream& fin, const string& varName, double& variable, const string& comment, bool text);

   Аргументы:

   ifstream& fin: ссылка на поток ввода, который используется для чтения данных из файла.

   const string& varName: имя переменной, которое может быть выведено для пользователя (если text установлено в true).

   double& variable: ссылка на переменную, в которую будет записано значение, считанное из файла.

   const string& comment: строка с комментарием, которая будет выведена после считывания (если не пустая).

   bool text: флаг, определяющий, нужно ли выводить имя переменной и ее значение на экран.

   Логика работы функции:

   Функция считывает значение переменной из файла с помощью потока fin и записывает его в переменную variable.

   Если флаг text установлен в true, то на экран выводится имя переменной varName и ее значение.

   Если строка comment не пустая, она выводится после считывания данных (может использоваться для добавления пояснений к выводимым данным).

   Пример вызова:

```
ifstream file("data.txt");
double value;
ReadDataFromFile(file, "Число", value, "Значение считано из файла", true);
```

2. **Перегруженная функция для чтения массива из открытого потока**:

   void ReadDataFromFile(ifstream& fin, const string& varName, double arr[], int size, const string& comment, bool text);

   Аргументы:

   ifstream& fin: ссылка на поток ввода для чтения данных из файла.

   const string& varName: имя массива, которое может быть выведено пользователю (если text установлено в true).

   double arr[]: массив типа double, в который будут записаны считанные данные.

   int size: размер массива, определяющий, сколько элементов нужно считать из файла.

   const string& comment: строка с комментарием, которая будет выведена после считывания данных.

   bool text: флаг, определяющий, нужно ли выводить имя массива и его элементы на экран.

Логика работы функции:

Функция последовательно считывает элементы массива из потока fin и сохраняет их в массив arr.

Если флаг text установлен в true, на экран выводится имя массива varName и каждый элемент массива.

По завершении вывода значений массива, если строка comment не пустая, она выводится для пояснения результата.

Пример вызова:

ifstream file("data.txt");

double values[5];

ReadDataFromFile(file, "Массив значений", values, 5, "Массив считан из файла", true);

Общие моменты для обеих версий функции:

Оба варианта функции принимают поток ifstream, из которого происходит чтение.

В зависимости от флага text, производится вывод информации о переменной или массиве.

Если строка comment передана и не пустая, она выводится для пояснения вывода или процесса считывания данных.

1.2.5. Функция WriteToFile для записи данных в файл: (не рекомендуется разный цвет шрифта)

void WriteToFile(const string& filename, const ostringstream& oss);

Аргументы:

const string& filename: строка, содержащая имя файла, в который будут записаны данные.

const ostringstream& oss: объект типа ostringstream, содержащий данные для записи в файл.

Логика работы функции:

ofstream fout(filename);

Открывается поток для записи в файл.

ofstream fout: создает объект fout типа ofstream для записи данных в файл.

filename: строка с именем файла, в который нужно записать данные. Если файл с таким именем не существует, он будет создан, если существует – данные будут перезаписаны.

string data = oss.str();

Содержимое объекта ostringstream преобразуется в строку.

oss.str(): функция str() возвращает данные, хранящиеся в объекте ostringstream, в виде строки.

Переменная data получает это строковое значение для дальнейшей записи в файл.

fout << data;

Запись данных в файл.

Оператор << используется для записи содержимого строки data в файл через поток fout. Весь текст, который был в объекте oss, будет записан в файл.

fout.close();

Закрытие файла.

После завершения записи данных в файл поток необходимо закрыть для освобождения ресурсов и завершения операции. Метод close() закрывает файл, связанный с потоком fout.

cout << "Данные записаны в файл " << filename << " в папке с программой.\n";

Сообщение пользователю о том, что данные успешно записаны.

После записи данных выводится сообщение в консоль с указанием имени файла. Пользователь получает подтверждение, что файл был создан или обновлен и что запись данных прошла успешно.

Пример вызова:

ostringstream outputStream;

outputStream << "Некоторые данные для записи";

WriteToFile("output.txt", outputStream);

Этот код создаст файл с именем output.txt в рабочей папке программы и запишет в него содержимое строки, хранящейся в объекте outputStream.

7.9.4. Отчет удаленного подчиненного программиста с титульным листом

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Удмуртский государственный университет» (ФГБОУ ВО «УдГУ»)
Кафедра информационной безопасности в управлении

Отчет подчиненного

Выполнил: подчиненный
студент группы ОБ-10.03.01.02-21 ФИО
Проверил: начальник ФИО

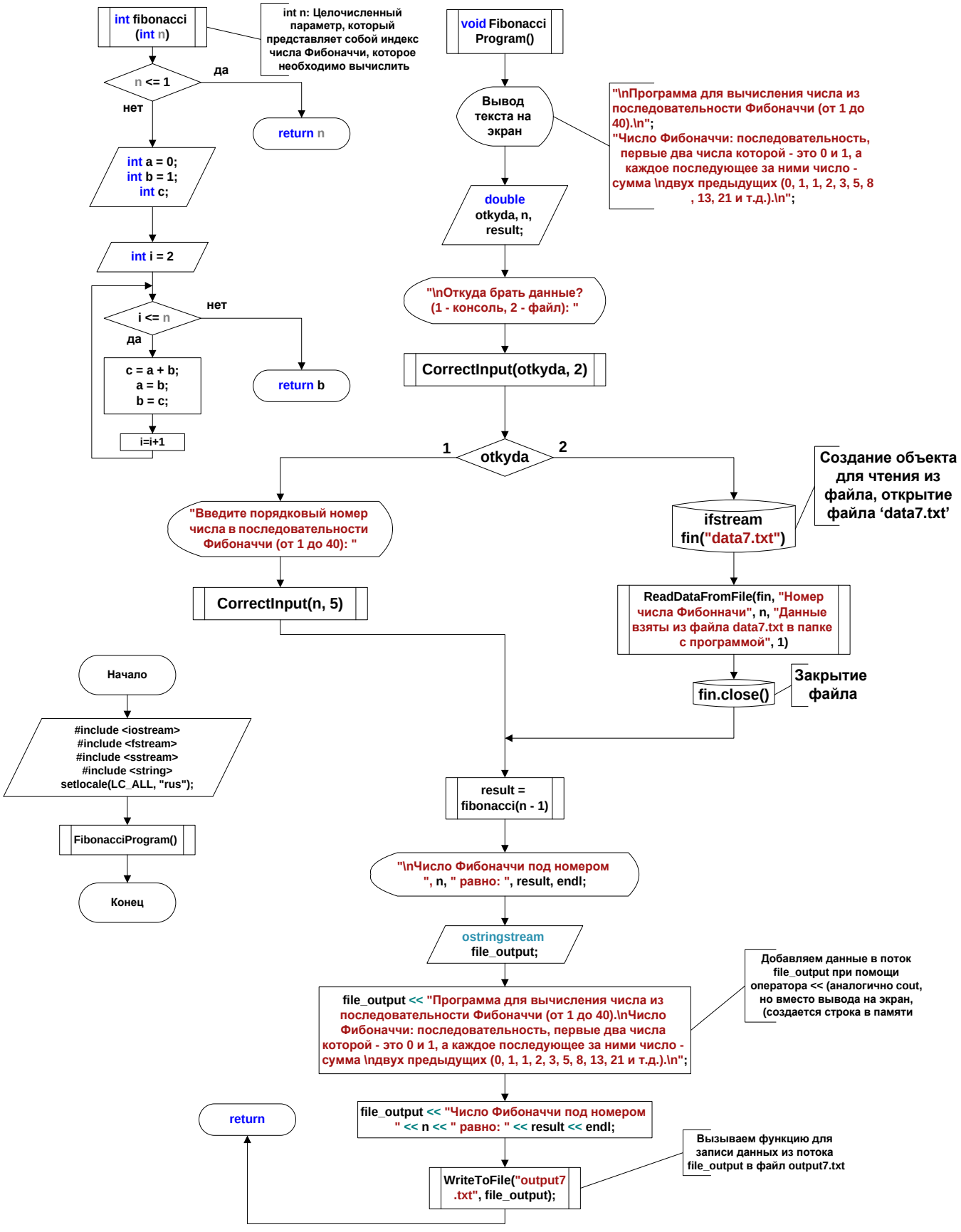
Ижевск 2026

Оглавление
(*нумерация страниц условно*)

Условие задачи

Написать программу согласно Внешней спецификации *(должна быть гиперссылка)* с функцией для вычисления числа из последовательности Фибоначчи. Аргументом функции является порядковый номер числа в последовательности.

Алгоритм решения в виде блок-схемы



Блок-схемы функций файлового ввода, записи в файл и защиты от некорректного ввода: гиперссылка – *не пронумерован рисунок*.

Текст программы на языке C++

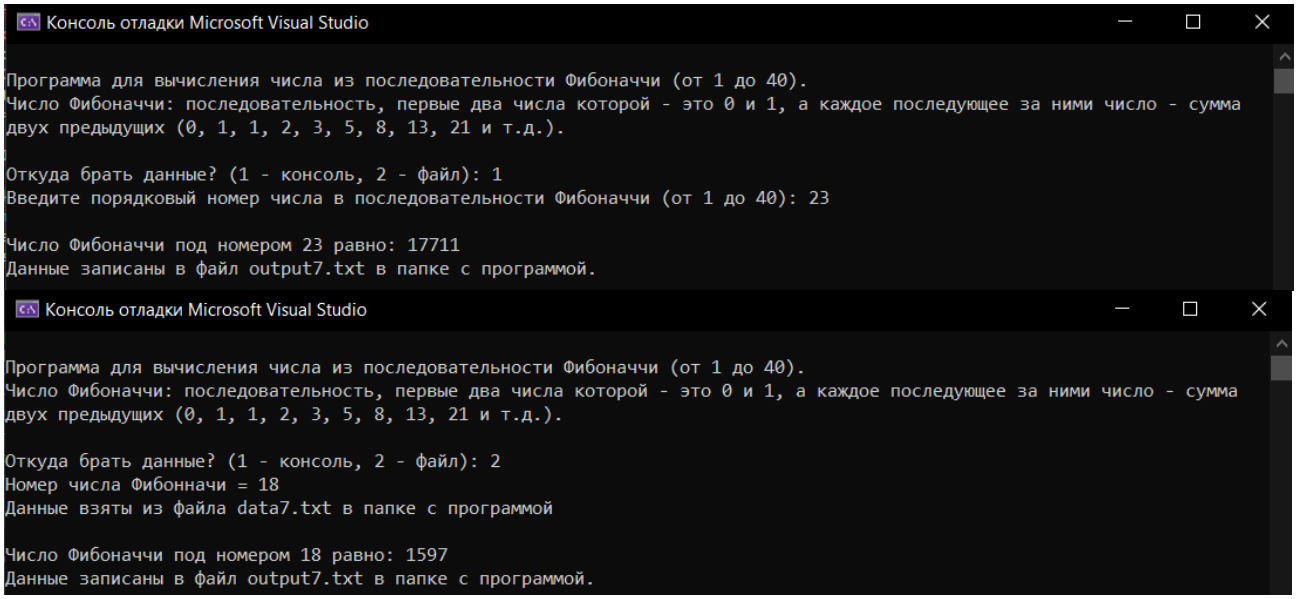
```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>

using namespace std;
// Функция проверки корректности ввода (&p1 – ссылка на считываемое значение, userflag – дополнительные условия проверки)
// (Защита от тупого, нелепого, наивного, придурковатого, простодушного, туповатого, безмозглого, легковерного, неосмысленного, непонимающего, глупого, недоумкуватого, неоднозначного, плоского, тупенького, бессмысленного, неумелого пользователя)
void CorrectInput(double& p1, int userflag) {
   string input;

   while (true) {
      getline(cin, input); // Считываем строку ввода
      // если встречается запятая, заменяем на точку
      for (size_t i = 0; i < input.size(); ++i)
         if (input[i] == ',')
            input[i] = '.';
      stringstream ss(input);
      double value;
      char extra;

      // Проверяем, является ли ввод корректным числом
      if (ss >> value && !(ss >> extra)) {
         // Проверяем дополнительные условия в зависимости от userflag
         if ((userflag == 1 && (value < 1 || value > 10000)) ||
            (userflag == 11 && ((value < 1 || value > 10000) || value != static_cast<int>(value))) ||
            (userflag == 2 && (value != 1 && value != 2)) ||
            (userflag == 3 && (value <= 0 || value > 100)) ||
            (userflag == 4 && (value < 0 || value > 1)) ||
            (userflag == 5 && (value < 1 || value > 40)) ||
            ((userflag == 3 || userflag == 5 || userflag == 0) && (value != static_cast<int>(value)))) {

            cout << "Некорректный ввод. Пожалуйста, попробуйте ввести данные заново." <<
endl;
         }
         else {
            p1 = value; // Присваиваем значение переменной p1
            break; // Ввод корректен, выходим из цикла
         }
      }
      else {
         // Если ввод некорректен (не число или содержатся лишние символы)
         cout << "Некорректный ввод. Пожалуйста, попробуйте ввести данные заново." <<
endl;
      }

      // Очищаем поток и сбрасываем ошибку
      cin.clear();
```

```
    }
}

// Перегруженная функция для чтения переменной из открытого потока
void ReadDataFromFile(ifstream& fin, const string& varName, double& variable, const string& comment, bool text) {
    fin >> variable;
    if (text)
        cout << varName << " = " << variable << endl;
    if (!comment.empty()) {
        cout << comment << endl;
    }
}

// Перегруженная функция для чтения массива из открытого потока
void ReadDataFromFile(ifstream& fin, const string& varName, double arr[], int size, const string& comment, bool text) {
    if (text) cout << varName << ": ";
    for (int i = 0; i < size; i++) {
        fin >> arr[i];
        if (text) cout << arr[i] << "  ";
    }
    cout << endl;
    if (!comment.empty()) {
        cout << comment << endl;
    }
}

// функция для записи данных в файл
void WriteToFile(const string& filename, const ostringstream& oss) {
    ofstream fout(filename);
    string data = oss.str();
    fout << data;
    fout.close();
    cout << "Данные записаны в файл " << filename << " в папке с программой.\n";
}


// Функция для вычисления числа Фибоначчи
int fibonacci(int n) {
    if (n <= 1) {
        return n; // Возвращаем n, если n меньше или равно 1
    }

    int a = 0; // Первое число Фибоначчи
    int b = 1; // Второе число Фибоначчи
    int c; // Для хранения следующего числа Фибоначчи

    for (int i = 2; i <= n; i++) {
        c = a + b; // Вычисляем следующее число
        a = b;     // Сдвигаем a и b
        b = c;     // b становится новым числом Фибоначчи
```

```
    }

    return b; // Возвращаем n-ное число Фибоначчи
}

// Программа 7: Числа Фибоначчи
void FibonacciProgram() {
    cout << "\nПрограмма для вычисления числа из последовательности Фибоначчи (от 1 до 40).\n";
    cout << "Число Фибоначчи: последовательность, первые два числа которой – это 0 и 1, а каждое последующее за ними число - сумма \nдвух предыдущих (0, 1, 1, 2, 3, 5, 8, 13, 21 и т.д.).\n";
    double otkyda, n, result;
    cout << "\nОткуда брать данные? (1 – консоль, 2 – файл): ";
    CorrectInput(otkyda, 2);
    if (otkyda == 1) {
        cout << "Введите порядковый номер числа в последовательности Фибоначчи (от 1 до 40): ";
        CorrectInput(n, 5);
    }
    else if (otkyda == 2) {
        ifstream fin("data7.txt");
        ReadDataFromFile(fin, "Номер числа Фибонначи", n, "Данные взяты из файла data7.txt в папке с программой", 1);
        fin.close(); // Закрытие файла
    }
    result = fibonacci(n - 1); // Вычисление числа Фибоначчи
    // При вызове функции `fibonacci(n - 1)` программа вычитает 1 из введённого пользователем номера, потому что последовательность Фибоначчи индексируется с 0. Например, если пользователь запрашивает первое число Фибоначчи (n = 1), это соответствует F(0), что равно 0. Таким образом, вычитание 1 позволяет правильно преобразовать пользовательский ввод в соответствующий индекс функции, обеспечивая корректный расчет числа Фибоначчи.
    cout << "\nЧисло Фибоначчи под номером " << n << " равно: " << result << endl;

    ostringstream file_output;
    file_output << "Программа для вычисления числа из последовательности Фибоначчи (от 1 до 40).\nЧисло Фибоначчи: последовательность, первые два числа которой - это 0 и 1, а каждое последующее за ними число - сумма \nдвух предыдущих (0, 1, 1, 2, 3, 5, 8, 13, 21 и т.д.).\n";
    file_output << "Число Фибоначчи под номером " << n << " равно: " << result << endl;
    WriteToFile("output7.txt", file_output);
}

// Главная функция
int main() {
    setlocale(LC_ALL, "rus");
    FibonacciProgram(); // Вызов основной программы
}
```

Для внедрения программы в общую, достаточно скопировать всё, кроме int main().

Результаты тестирования программы



Вывод:

В процессе создания подпрограммы для вычисления ряда были соблюдены все требования Внешней спецификации. Код рабочий и подготовлен в виде подпрограмм int fibonacci(int n) и void FibonacciProgram() для вставки в основную часть программы. Начальник работу исполнителя принял и проверил.

8. Варианты заданий по лабораторной работе № 2

Варианты заданий по лабораторной работе № 2 – это лабораторные работы № 7–10 из предыдущего семестра. Лабораторная работа № 2 служит для закрепления знаний при создании технологического процесса разработки программного продукта и включает в себя такой же перечень документов кроме п. 7.4. Модель компании – разработчика ПО становится меньше на двух кодировщиков и удаленного программиста.

9. Требования к защите лабораторной работы

Защита лабораторной работы состоит из демонстрации работы программы в соответствии с указанными ранее требованиями и правильно составленного отчета в формате MS Word (комплекта документов). После защиты работы отчет сдается преподавателю в электронном виде в формате Word. В случае искажений форм рисунков отчета дополнительно предоставляется копия отчета в формате PDF.

Список литературы

1. Васильев, А. Н. Самоучитель C++ с примерами и задачами / под ред. М. В. Финкова. − 6-е изд. (перераб. и обновленное) [Книга + виртуальный CD]. – Санкт Петербург : Наука и Техника, 2019. − 480 с.

2. ГОСТ 19.201-78. Группа Т55. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению. МКС 35.080 (принят постановлением Государственного комитета СССР по стандартам от 18 декабря 1978 г. № 3351, введен 01.01.80 с Изменением № 1, утвержденным в июне 1981 г. (ИУС 9-81)) // Электронный фонд правовых и нормативно-технических документов : [сайт]. – URL: https://docs.cntd.ru/document/1200007648 (дата обращения: 22.04.206).

3. ГОСТ 19.102-77. Группа Т55. Межгосударственный стандарт. Единая система программной документации. Стадии разработки. МКС 35.080 (принят постановлением Государственного комитета стандартов Совета Министров СССР от 20 мая 1977 г. № 1268 введен 01.01.80. Переиздание. Январь 2010 г. // Электронный фонд правовых и нормативно-технических документов : [сайт]. – URL: https://docs.cntd.ru/document/1200007628 (дата обращения: 22.04.206).

4. ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств. ОКС 35.080 (утвержден и введен в действие приказом Федерального агентства по техническому регулированию и метрологии от 30 ноября 2010 г. № 631-ст) // Электронный фонд правовых и нормативно-технических документов : [сайт]. – URL: https://docs.cntd.ru/document/120008285 (дата обращения: 22.04.206).

5. Павлов, Ф. Ф. Технология разработки программного обеспечения : учебное пособие для СПО / И. Г. Гниденко. Ф. Ф. Павлов, Д. Ю. Федоров. – Москва : Юрайт, 2019. – 235 с. – (Серия: Профессиональное образование).

6. Ларина, Н. А. Технологии программирования : учебное пособие для бакалавров направления «Информатика и вычислительная техника» / Рубцовский индустриальный институт. – Рубцовск, 2016. – 51 с.

7. Гайсин, М. М. Методика выполнения лабораторных работ по дисциплине «Языки программирования» : методические рекомендации. – Ижевск : Удмуртский университет, 2022. – 28 с.

8. Фаулер, М. UML. Основы. – 3-е изд. – Пер. с англ. – Санкт-Петербург : Символ-Плюс, 2004. – 192 с., ил. ISBN 5-93286-060-X.

ОПИСАНИЕ ФУНКЦИОНАЛЬНОСТИ ИЗДАНИЯ:

Интерфейс электронного издания (в формате pdf) можно условно разделить на 2 части.

Левая навигационная часть (закладки) включает в себя содержание книги с возможностью перехода к тексту соответствующей главы по левому щелчку компьютерной мыши.

Центральная часть отображает содержание текущего раздела. В тексте могут использоваться ссылки, позволяющие более подробно раскрыть содержание некоторых понятий.

МИНИМАЛЬНЫЕ СИСТЕМНЫЕ ТРЕБОВАНИЯ:

Celeron 1600 Mhz; 128 Мб RAM; Windows XP/7/8 и выше; 8x CD-ROM; разрешение экрана 1024×768 или выше; программа для просмотра pdf.

СВЕДЕНИЯ О ЛИЦАХ, ОСУЩЕСТВЛЯВШИХ ТЕХНИЧЕСКУЮ ОБРАБОТКУ И ПОДГОТОВКУ МАТЕРИАЛОВ:

Оформление электронного издания : Издательский центр «Удмуртский университет».

Компьютерная верстка: И. А. Бусоргина

Авторская редакция

Подписано к использованию 17.09.2026

Объем электронного издания 1,6 Мб

Издательский центр «Удмуртский университет»

426034, г. Ижевск, ул. Ломоносова, д. 4Б, каб. 021

Тел. : +7(3412)263-751 E-mail: editorial@udsu.ru