Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный
университет имени П.М. Машерова»
Кафедра прикладного и системного программирования

С.А. Ермоченко, Т.К. Петрова

ИНСТРУМЕНТАРИЙ РАЗРАБОТЧИКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические рекомендации

Витебск
ВГУ имени П.М. Машерова
2022

УДК 004.4’2(075.8)
ББК 32.972я73
Е74

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 2 от 05.01.2022.

Авторы: заведующий кафедрой прикладного и системного программирования ВГУ имени П.М. Машерова, кандидат физико-математических наук, доцент **С.А. Ермоченко;** старший преподаватель кафедры прикладного и системного программирования ВГУ имени П.М. Машерова **Т.К. Петрова**

Р е ц е н з е н т :
заведующий кафедрой «Информационные системы
и автоматизация производства» УО «ВГТУ»,
кандидат технических наук, доцент *В.Е. Казаков*

Ермоченко, С.А.

Е74 Инструментарий разработчика программного обеспечения : методические рекомендации / С.А. Ермоченко, Т.К. Петрова. – Витебск : ВГУ имени П.М. Машерова, 2022. – 52 с.

В методических рекомендациях изложены основные принципы применения таких инструментов, как системы управления проектами, офисные приложения, консоль, системы управления версиями, системы запуска модульных тестов.

Данное издание предназначается для студентов первой ступени высшего образования специальностей: «Управление информационными ресурсами» (дисциплина «Системная интеграция и конфигурирование программного обеспечения»); «Информационные системы и технологии (в здравоохранении)» (дисциплина «Технологии разработки программного обеспечения»); «Программное обеспечение информационных технологий» (дисциплина «Основы программной инженерии»); «Прикладная информатика» (дисциплина «Введение в специальность»).

УДК 004.4’2(075.8)
ББК 32.972я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современная разработка программного обеспечения – это сложный процесс, в котором задействуются много специалистов разного профиля. Для организации удобного взаимодействия между членами команды, работающей над одним проектом, применяются различные программные инструменты.

Владение этим инструментарием является важной частью навыков, требуемых для работы программиста.

В данном методическом пособии рассматриваются инструменты, применяемые практически на всех этапах жизненного цикла программного обеспечения.

На этапе анализа и документирования требований применяются обычные офисные приложения, системы управления проектами и Wiki-страницы.

На этапе проектирования и написания программного кода применяются системы контроля версий.

На этапе внедрения проектов применяются различные консольные утилиты, для работы с которыми важны навыки работы в консоли.

Материал соответствует темам учебных программ курсов: «Системная интеграция и конфигурирование программного обеспечения» (специальность «Управление информационными ресурсами»); «Технологии разработки программного обеспечения» (специальность «Информационные системы и технологии (в здравоохранении)»); «Основы программной инженерии» (специальность «Программное обеспечение информационных технологий»); «Введение в специальность» (специальность «Прикладная информатика»).

1. ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Жизненный цикл – это период времени между моментом принятия решения о разработке программного продукта до момента полного вывода его из эксплуатации.

Стоит отметить, что момент принятия решения – это некоторое подтверждённое каким-либо образом событие. Например, подписание договора с заказчиком или устное соглашение о необходимости разработать некоторое программное обеспечение.

Жизненный цикл программного обеспечения состоит из определённых этапов, на каждом из которых решается некоторый определённый набор задач, и результат которого передаётся на следующий этап в виде некоторых артефактов: документ, исходный код, база данных и т.д.

Традиционно жизненный цикл состоит из следующих пяти этапов.

1. Анализ.
2. Проектирование.
3. Разработка.
4. Тестирование.
5. Внедрение.

В реальности в зависимости от специфики разрабатываемого программного продукта количество и назначение этапов могут меняться. Так же последовательность этих этапов может меняться и вообще быть нелинейной. Это зависит от выбранной методологии управления проектами. Строго последовательно друг за другом этапы идут только в каскадной методологии, которая как раз в последнее время применяется не так часто.

В рамках данных рекомендаций мы рассмотрим только цель каждого этапа, ожидаемый результат и участников (тех специалистов, работа которых на каждом этапе имеет важное значение). Участниками этапа являются специалисты, которые могут решать различные задачи и на других этапах. В таком случае говорят о роли или о нескольких ролях, которые специалист может занимать на проекте.

1.1. Анализ требований

***Цель этапа*:** сбор, анализ и документация требований к программному обеспечению.

Ожидаемый результат: описание требований в виде некоторых формализованных документов (техническое задание, vision and scope, requirement specification и т.д.); набор требований в специальных системах управления требованиями (RMS – Requirement Management Systems); прототип.

Прототипирование – это способ, позволяющий с большей точностью понять ожидания заказчика, проверить выполнимость некоторых возникающих идей, оценить графический пользовательский интерфейс разрабатываемого ПО.

В основном прототип – это некоторая сильно ограниченная по своим функциональным возможностям версия программного продукта, но создание которой может быть выполнено в относительно короткие сроки (от 2–3 недель до 1–2 дней). Зачастую задача прототипа – просто предоставить заказчику визуализацию интерфейса. В этом случае прототип может даже быть не кликабельным, то есть представлять собой статическую картинку. Кликабельный прототип позволяет с ним некоторым образом взаимодействовать, но при этом фактически он может не реализовать функционал, а лишь его имитировать, помогая понять, как в будущем этот функционал будет работать.

Некликабельный прототип может достаточно подробно показывать внешний вид страниц, форм или экранов будущего приложения. Этот прототип в последующем берётся за основу разработчиками, которые будут реализовывать интерфейс пользователя. Такой прототип называется дизайн-макетом.

Но прототип может показывать внешний вид приложения и условно, схематически. Такие простейшие прототипы могут просто рисоваться на листе бумаги или на белой маркерной доске во время интервью или мозговых штурмов.

***Участники этапа*:**

Бизнес-аналитик (Business Analyst) – специалист, выполняющий непосредственный сбор и анализ требований заказчика и их последующее документирование. Его задача – собрать требования у заинтересованных лиц, проанализировать цели создания программного продукта, сформулировать для заказчика свои предложения. Зачастую бизнес-аналитик – это тот, кто должен понять, что действительно нужно заказчику, а не то, что заказчик просит сделать, и объяснить это заказчику. В дальнейшем, когда бизнес-аналитик согласовал требования с заказчиком, он документирует требования на понятном для разработчиков языке, и играет роль посредника между командой разработчиков и представителями заказчика.

Дизайнер интерфейсов (UI Designer) – специалист, отвечающий за создание удобного графического интерфейса пользователя. На этапе анализа требований привлекается к созданию прототипа, помогая лучше понять ожидания заказчика. Поскольку задача дизайнера – создание удобного, а не просто стильного и красивого интерфейса, то ему необходимо изучить фокус-группу, то есть круг будущих пользователей программного продукта, и исходя из их особенностей спроектировать интерфейс приложения. При этом управляющие элементы на каждой странице / экране / форме должны быть расположены оптимальным образом, так, чтобы наиболее часто нужные пользователю действия можно было осуществить максимально просто и быстро. А также оптимизированы должны быть переходы между страницами / экранами / формами, так, чтобы попасть на наиболее востребованные

можно было максимально удобно. Таким образом, основные профессиональные компетенции дизайнера интерфейсов – это понимание специфики работы пользователей с теми или иными данными и функциями в приложениях, а не умение хорошо рисовать и пользоваться графическими редакторами. Хотя, конечно, последнее тоже важно для дизайнера.

1.2. Проектирование архитектуры

***Цель этапа*:** Планирование и проектирование внутренней структуры программного продукта. Создание так называемой архитектуры.

***Ожидаемый результат*:** Набор диаграмм и документов, показывающих созданную архитектуру ПО.

Зачастую разработанная архитектура должна помогать разработчиками максимально быстро и гибко вносить необходимые изменения в исходный код программы в случае изменения требований или необходимости исправить обнаруженные дефекты. Для этого любой разработчик должен иметь возможность максимально быстро вникать в исходный код, написанный другими разработчиками, или свой же собственный исходный код, написанной ранее. Поэтому архитектура приложения должна быть унифицирована и хорошо структурирована. Взаимодействие отдельных частей или модулей приложения должно быть чётко описано и позволять гибко заменять или добавлять новые модули.

Также должна быть описана структура постоянного хранилища данных, обрабатываемых приложением. Эта структура также должна позволять быстро понять, где нужно внести какое изменение в случае необходимости. Но также структура приложения должна быть оптимизирована с точки зрения скорости чтения, добавления, модификации и удаления данных, а также с точки зрения возможности хранения необходимых заказчику объёмов данных.

***Участники этапа*:**

Системный архитектор (System Architect) – специалист по проектированию архитектуры систем, при этом он планирует и разрабатывает не только внутреннюю архитектуру программного кода системы, но и аппаратную архитектуру системы. Важной особенностью системного архитектора является понимание требований и проблем заказчика и создание решений, ориентированных на его потребности. Как правило, это опытный сотрудник, ранее занимавшийся непосредственно разработкой программ и / или сопровождением работы сложных систем.

Системный аналитик (System Analyst) – специалист по анализу имеющихся и проектированию новый сложных систем. Его основное отличие от системного архитектора в том, что при анализе систем он использует строгий математический аппарат теории системного анализа. По спектру решаемых задач занимает промежуточное положение между бизнес-аналитиком и системным архитектором.

Администратор баз данных (DataBase Administrator) – специалист по проектированию, разработке и поддержке работы постоянных хранилищ данных (баз данных). Его задача – создание и оптимизация баз данных. Поддержка работы баз данных: обеспечение надёжности и отказоустойчивости подсистемы хранения, настройка распределённых подсистем хранения, обеспечение резервного копирования баз данных и восстановления после сбоев. Также тесно работает с разработчиками, обеспечивая взаимодействие разрабатываемого программного продукта с подсистемой хранения данных.

Инженер по обработке данных (Data Engineer) – фактически, это эволюция роли администратора баз данных. Так как в настоящее время всё больше и больше программных систем обрабатывают данные, которые хранятся не только в самой системе заказчика, и на сторонних сервисах или в облачных хранилищах, то подсистема сбора, хранения и обработки данных становится очень сложной и размытой. Задача данного специалиста обеспечить единый доступ ко всем необходимым данным, сделать их доступность максимально простой для разработчиков. В отличие от администратора баз данных инженер по обработке данных не занимается сервисным обслуживанием хранилищ, то есть оптимизацией производительности, поддержкой отказоустойчивости и надёжности и т.д. Его основная задача – это организация сбора и обработки данных из различных источников, унификация этих данных и настройка так называемых Pipeline для обработки данных, т.е. этапов обработки данных и передачи данных между узлами системы обработки данных.

Аналитик данных (Data Scientist) – это специалист, который занимается обработкой данных с помощью различных математических методов. Его основная цель – извлечь пользу для заказчика из имеющихся данных, обнаружить некоторые скрытые закономерности, анализирует внутренние процессы заказчика и выявляет потребность в данных для управления этими процессами, визуализирует данные, подготавливает отчёты на основе имеющихся данных. Фактически его задача – обработать те данные, которые подготовил для него инженер по обработке данных, спроектировать алгоритмы обработки этих данных и предоставить их разработчикам, чтобы они реализовали эти алгоритмы в приложении, интегрировав их в бизнес-процессы заказчика.

Специалист по машинному обучению (Machine Learning Engineer) – эту специальность можно рассматривать как частный случай аналитика данных, у которого специализация на конкретных методах обработки данных. Это прежде всего искусственные нейронные сети. А также другие методы машинного обучения, которые могут помочь на основе имеющихся данных принять правильное управленческое решение за человека в случаях, когда алгоритмы обработки исходных данных не могут быть строго формализованы или, когда такая формализация может занять неприемлемое время.

Бизнес-аналитик данных (Business Intelligence) – это ещё один частный случай аналитика данных, у которого акцент смещён с методов обработки

данных на цель обработки данных. Фокус этого специалиста размещён на потребностях бизнес-процессов заказчика и простоте восприятия данных. В основном задача такого специалиста – предоставить человеку, принимающему решение, выборку из имеющегося объёма данных, исключив вспомогательную и несущественную информацию и представив нужную информацию в максимально удобном для анализа и принятия решения виде.

1.3. Разработка

Цель этапа: Непосредственная реализация программного продукта в виде работающей системы.

Ожидаемый результат: Исходный код программной системы, который можно развернуть и запустить на оборудовании заказчика.

Данный этап обычно называют основным этапом жизненного цикла программного обеспечения. Но ранжировать этапы по важности или ценности для всего жизненного цикла – неверно. Можно говорить о том, что этом один из самых длинных этапов жизненного цикла. Но если вспомнить, что жизненный цикл программного продукта заканчивается только после полного вывода системы из эксплуатации, то самым длинным этапом является последний – этап внедрения. Можно сказать, что этап разработки – это этап с максимальным удельным временем работы IT-специалистов.

Для того, чтобы выделить возможных участников этого этапа, необходимо рассмотреть, какие виды программных продуктов могут создаваться разработчиками. Из наиболее популярных видов приложений рассмотрим следующие:

– веб-приложения;

– приложения для мобильных платформ;

– настольные приложения;

– приложения для встраиваемых систем.

Веб-приложения характеризуются своей распределённостью. Основная часть работает на некотором сервере. Эту часть называют Back-End. И может присутствовать вторая часть, работающая в браузере пользователя. Эту часть называют клиентской, или Front-End.

Существует большое количество языков программирования и программных платформ, на которых можно разрабатывать Back-End. Из наиболее популярных можно назвать Java, .Net, Python, Node.js, PHP, C/C++.

Разработка Front-End для веб-приложений в основном ведётся с помощью языков разметки HTML и CSS, а также языка программирования JavaScript или его ближайших аналогов TypeScript и CofeeScript.

Приложения для мобильных платформ в основном делятся по самим платформам Android и iOS. Также выделяют отдельное направление кроссплатформенной разработки под мобильные платформы, например, с помощью библиотеки React Native и языка программирования JavaScript или TypeScript.

Настольные приложения также делятся по платформам (Windows, MacOS) или могут быть кроссплатформенными.

Приложения для встраиваемых систем – это приложения, исполняемые на специальных устройствах. Например, в бытовых микроволновых печах или в стиральных машинах, в устройствах, которые относятся к классу «Интернета вещей» и т.п. Основной особенностью таких систем является отсутствие полноценного процессора, вместо которого используются различного вида микроконтроллеры, ограниченные по функционалу.

При этом и мобильные, и настольные, и встраиваемые приложения могут быть лишь частью более общей системы и взаимодействовать с Back-End частью с помощью специально разработанного программного интерфейса (API – Application Programming Interface).

Участники этапа:

Разработчик серверной части (Back-End Developer) – специалист, имеющий, как правило, некоторую специализацию по используемому языку программирования или платформе (Java-разработчик, .Net-разработчик, Python-разработчик, и т.д.), отвечающий за разработку Back-End части системы. Как правило выполняет обработку данных, хранящихся в базе данных или других источниках данных. Реализует основную бизнес-логику приложения.

Разработчик клиентской части (Front-End Developer) – специалист, отвечающий за разработку Front-End части веб-приложений. Как правило выполняет визуализацию данных и обеспечивает взаимодействие пользователя с системой.

Разработчик и серверной, и клиентской части (Full Stack Developer) – специалист, совмещающий в себе должность Back-End и Front-End разработчика. К такому специалисту предъявляется больше требований, но и оплата его труда выше, решаемый круг задач разнообразнее и интереснее. С точки зрения целей проекта такой специалист зачастую более выгоден, так как лучше понимает работу всей системы в целом и менее ограничен в выборе средств для реализации той или иной задачи.

Разработчик настольного ПО (Desktop Developer) и разработчик ПО для мобильных платформ (Mobile Developer) – специалист по созданию ПО для некоторой конкретной платформы, или некоторого множества платформ (в случае кроссплатформенной разработки). Фактически, как и Front-End разработчик, отвечает за визуализацию данных и обеспечение взаимодействия пользователя с системой. Но может использовать различные особенности платформы для более эффективного достижения результата. Может глубже оптимизировать работу системы на стороне клиента. В случае разработки толстого клиента (это клиент, выполняющий обработку данных самостоятельно, без сервера, в таких случаях сервер отвечает только за сбор и хранение данных, зачастую на сервере может работать только система управления базами данных без отдельной Back-End части) разрабатывает основной функционал бизнес-логики.

Разработчик встраиваемого программного обеспечения (Embedded Software Developer) – специалист по разработке приложений в условиях довольно жёстких ограничений целевой встраиваемой системы. Как правило должен разбираться в системных особенностях встраиваемых систем, иметь опыт оптимизации исходного кода по различным критериям. Работает с так называемым низкоуровневым программированием.

1.4. Тестирование

Цель этапа: Обеспечение качества разрабатываемого продукта.

Ожидаемый результат: После выполнения некоторого набора тестов подготавливается отчёт о дефектах (Bug Report) и рекомендации для разработчиков. Эти отчёты и рекомендации должны помочь разработчикам устранить дефекты и способствовать уменьшению появления новых дефектов в будущем.

Важной особенностью данного этапа является отсутствие гарантии отсутствия дефектов. Специалисты по тестированию могут гарантировать, что в том объёме, в котором они провели тестирование системы, дефекты не были выявлены. Но гарантировать полного отсутствия дефектов специалисты по тестированию не могут.

Участники этапа:

Специалист по ручному тестированию (Manual Tester) – специалист, планирующий процесс тестирования, выполняющий запланированные тесты вручную в нужном объёме и составляющий отчёты об обнаруженных дефектах.

Специалист по автоматизированному тестированию (Automation Tester) – специалист, автоматизирующий выполнение тестов, которые часто приходится выполнять специалистам по ручному тестированию. Автоматизация заключается в написании программного кода, который имитирует поведение пользователя в том или ином случае, и проверяет корректность работы системы (не отличается ли фактическое поведение системы от ожидаемого).

Специалист по нагрузочному тестированию (Load Tester) – эту специализацию можно, как и специалиста по тестированию безопасности (Security Tester), считать частным случаем специалиста по ручному или автоматизированному тестированию. Однако эти специалисты должны обладать рядом специфичных компетенций, связанных с глубоким пониманием внутренней особенности тестируемой системы. Специалист по нагрузочному тестированию проверяет, справится ли система с тем объёмом работ, который необходим заказчику. Его задача протестировать, какое время система будет обрабатывать те или иные запросы пользователей, как система будет себя вести, если количество пользовательских запросов будет со временем возрастать и т.д. Специалист по тестированию безопасности проверяет устойчивость системы к различным попыткам незаконного проникновения в систему.

1.5. Внедрение

Цель этапа: Развернуть разработанную систему на оборудовании заказчика и обеспечить её стабильную работу в реальных условиях.

Ожидаемый результат: Работающая система, отвечающая всем ожиданиям заказчика и конечных пользователей.

Данный этап может быть самым разнообразным по способам решения задач. Всё зависит от специфики разработанной системы.

Например, если разрабатывалось только лишь некоторое мобильное приложение, то достаточно разместить приложение в некотором публичном репозитории (например, в Google Play Market для платформы Android). После чего необходимо лишь выполнять мониторинг обратной связи пользователей и выпускать обновления приложения с исправленными проблемами. А в некоторых случаях может понадобиться целый комплекс работ по разворачиванию системы в предоставленной заказчиком инфраструктуре предприятия с задействованием как серверов самого заказчика, так и некоторой облачной платформы. А затем обеспечить поддержку работу данной системы в режиме 24/7.

Именно поэтому данный этап могут называть по-разному: внедрение, сопровождение, сдача в эксплуатацию. В первом случае команда разработчиков осуществляет разворачивание системы, но дальнейшая эксплуатация осуществляется самим заказчиком. Во втором случае команда разработчиков сопровождает систему в течение определённого времени после внедрения. Это время может быть достаточно значительным, вплоть до полного изъятия системы из эксплуатации. В третьем случае и сопровождение, и даже внедрение системы берёт на себя сам заказчик. Команда разработчиков лишь осуществляет передачу необходимых артефактов заказчику.

В случае сопровождения работы системы отдельно решается задача технической поддержки конечных пользователей, которая может организовано в несколько уровней, которые часто называются линиями. Например, первая линия технической поддержки принимает все обращения от пользователей и оказывает консультационную поддержку, объясняя пользователям, что они делают неправильно в системе, и помогая совершить нужные действия для получения ожидаемого результата. Как правило, на первой линии технической поддержки работают неквалифицированные специалисты, не имеющие технических знаний. И только если специалист первой линии технической поддержки понимает, что проблема заключается не в неправильных действиях пользователя, а в неправильной работе самой системы, этот инцидент передаётся на вторую линию технической поддержки, где работают более квалифицированные специалисты, способные решить эту проблему на уровне самой системы.

Участники этапа:

Системный администратор (System Engineer) – специалист, осуществляющий непосредственную настройку и поддержку работы системы.

Его можно назвать специалистом последней линии технической поддержки. Но даже он не способен исправить ошибки в коде самой программной системы. Он может лишь передать информацию об этих ошибках разработчикам, максимально детализировав эту информацию техническими подробностями. В круг задач этого специалиста входит мониторинг поведения системы, объёма потребляемых ресурсов (загрузка вычислительных мощностей системы, объём используемой оперативной и постоянной памяти, объём используемого сетевого трафика и т.д.). Реагирование на нестандартные ситуации: программные и аппаратные сбои, попытки несанкционированного доступа, отсечение нежелательного трафика и т.д. Выполнение сервисных операций: резервное копирование данных и подсистем, очистка временных хранилищ от уже ненужной информации, периодическое обновление используемого программного обеспечения.

Специалист по доставке программного обеспечения (DevOps Engineer) – специалист, совмещающий в себе роль системного администратора и разработчика, отвечающего за разворачивание готовой системы. Строго говоря, отдельных специалистов в области DevOps нет. Это те же системные инженеры, но применяющие практики DevOps, такие как непрерывная интеграция и непрерывная доставка программного обеспечения, обеспечение взаимодействия разработанной системы с облачными сервисами. Основная задача таких специалистов обеспечить максимально быструю доставку результатов одного этапа специалистам другого этапа или заказчику. Например, если разработчики подготовили очередную версию (сборку) программной системы, так называемый релиз, задача DevOps инженеров настроить автоматическую доставку обновлённой версии специалистам по тестированию. После того, как разработчики проверили новый релиз о размещают его в специальном общем месте (в некоторой специальной части центрального репозитория исходного кода), этот релиз автоматически разворачивается в специальном тестовом окружении, подготовленным для специалистов по тестированию. После того, как разворачивание будет завершено, специалисты по тестированию получают уведомление о том, что новый релиз готов для тестирования. При этом тесты, созданные специалистами по автоматизированному тестированию, могут быть запущены автоматически после разворачивания очередного релиза. При этом специалисты по автоматизированному тестированию могут заранее настроить, какие тесты должны быть запущены в конкретном релизе. Весь процесс может занять всего пару часов. Также автоматически релиз может быть доставлен пользователям, когда успешно протестированный релиз будет автоматически размещён в другой специальной части центрального репозитория исходного кода. Все эти правила обработки и доставки разработанного программного обеспечения определяются системными инженерами, применяющими практики DevOps.

1.6. Практическое задание – эссе

Составьте краткое эссе на тему «***В какой проектной роли я вижу себя после окончания университета***».

В эссе необходимо аргументировать выбор проектной роли (или нескольких ролей, в этом случае желательно указать приоритет этих ролей лично для Вас). Для чего необходимо проанализировать свои предпочтения, склонности, способности и черты характера и сопоставить их с требованиями к существующим ролям. Можно дополнить эссе другими ролями, не рассмотренными выше, например, специалистами из области управления проектами, или специалистами из области разработки компьютерных игр (GameDev).

Требования к эссе: создать документ на Google Диске, размер листа А4, поля по краям страницы 1,5 см, шрифт Verdana размером 14 pt. Отступы вокруг абзаца 0 см. Отступ красной строки 1 см. Выравнивание абзаца по ширине. Междустрочный интервал 1,5 строки. Минимальный объём 250 слов. Максимальный объём 2 страницы.

Предоставить преподавателю доступ на чтение к этому документу.

2. СИСТЕМЫ УПРАВЛЕНИЯ ТРЕБОВАНИЯМИ

На этапе формирования и анализа требования решаются следующие основные задачи:

– сбор требований;

– анализ требований;

– документирование требований.

Управление требованиями на этапе анализа требований и дальнейшая работа с требованиями могут осуществляться в специализированных системах управления требований. Подавляющее большинство таких систем являются коммерческими разработками и ориентированы прежде всего на сложные программные продукты с неочевидными требованиями.

При реализации некоторого конкретного требования разработчикам ставятся задачи. При этом на одно требование может создаваться несколько разных задач для разных разработчиков. У каждой такой задачи в рамках одного требования может быть свой приоритет, своё время выполнения, своя сложность. Но на практике часто большая часть задач сводится к формулировке «Реализовать требование #1234». Поэтому в последнее время зачастую требование к продукту и задачу для разработчика уже не различают. Особенно в рамках популярных гибких методологий Scrum, Kanban или Extreme Programming, которые стараются свести объём документации к минимуму. В таком случае задача управления требованиями сводится к задаче управления задачами на проекте.

В качестве систем управления задачами используются или системы управления проектами, в которых можно формировать список задач, разбивая их на подзадачи, назначая задачи разным разработчикам и отслеживая статус выполнения каждой задачи. К таким системам относятся, например, Jira или Битрикс24.

В случае же, когда разбиение задач на подзадачи не требуется, достаточно формировать список задач с различными метками и статусами, назначать задачи разным разработчикам, зачастую используются системы управления дефектами (Bug Tracking Systems), например, система BugZilla. При этом системы управления дефектами оперируют понятием дефекта, в котором есть поля, аналогичные задаче: описание, статус, исполнитель, метки. Объединение понятия задача и дефекта (во многих системах для них используется один термин Issue) упрощает процесс документации проекта.

В настоящее время популярными являются системы, которые объединяют в себе системы контроля версий (чаще всего GIT), системы управления дефектами / задачами, системы непрерывной интеграции и доставки (CI/CD – Continuous Integration / Continuous Delivery) и технологию Wiki-страниц. Примерами таких систем являются GitHub, GitLab, Bitbacket и т.д. В таких системах набор связанных Wiki-страниц используют для документирования проекта, в том числе и требований. Но управлять требованиями в таком формате неудобно. Управление осуществляется задачами (или Issues), которые хранят ссылки на соответствующие требования.

2.1. Сбор требований

При сборе требований основными путями выявления требований являются следующие.

1. Нормативные документы.

В случае, если разрабатываемый программный продукт автоматизирует или моделирует бизнес-процессы, которые должны подчиняться некоторым законодательным актам, локальным положениям, приказам и регламентам организации-заказчика, тогда основным источником требований становятся эти документы.

Анализ нормативно-правовой документации имеет несколько плюсов. Во-первых, такие документы обладают высокой степенью формализации. Во-вторых, анализ документа не требует от бизнес-аналитика постоянного взаимодействия с заказчиком. Последнее часто может быть затруднено занятостью сотрудников организации-заказчика. В-третьих, опытные бизнес-аналитики, уже работавшие в сфере, в которой выполняется некоторый текущий проект, уже, как правило, достаточно хорошо знают и сами нормативные документы, и некоторые особенности их применения на практике, что существенно экономит время на стадии бизнес-анализа.

Однако необходимость следовать требованиям нормативных документов может иметь и несколько негативных последствий. Во-первых, чрез-

мерно высокая степень формализации тех или иных процессов может требовать определённых действий, которых при автоматизации с помощью информационных технологий можно было бы избежать, однако наличие этих формальных требований заставляют усложнять бизнес-процессы, что делает менее прозрачными и затрудняет разработку системы. Во-вторых, в нормативных документах тоже могут присутствовать нестыковки и противоречия с другими документами или даже внутри одного документа. Если такие противоречие возникают во внутренних документах организации-заказчика, это можно исправив, внеся изменения в эти документы. Это может немного замедлить процесс бизнес-анализа, но такие внутренние противоречия помогают оптимизировать сами бизнес-процессы, упрощая не только разработку программного продукта, но и действия конечных пользователей программного продукта за рамками использования разрабатываемой системы. Однако, когда противоречия обнаруживаются в законодательных актах, изменить которые нет никакой возможности, приходится искать пути обхода спорной ситуации и принимать меры по её недопущению.

2. Возможности аналогов. Когда разрабатываемый программный продукт уже имеет некий аналог, то большую часть функционала можно не описывать в требованиях, а ограничиться лишь требуемыми отличиями и нюансами.

3. Ожидания пользователей. С одной стороны – самый естественный способ сбора требований, но и самый трудоёмкий и неоднозначный. Как правило – программный продукт – это многопользовательская сложная система с распределением различных ролей. В таком случае практически невозможно опросить всех пользователей, которые будут работать с приложением, и уж тем более учесть все их пожелания. Поэтому такой способ выявления и сбора требований сопряжён с постоянным поиском компромиссов и риском постоянных изменений требований к проекту. Однако вообще не учитывать мнение пользователей – тоже не правильный подход. Как правило, бизнес-аналитик должен найти ту золотую середину, которая позволит ему собрать все требования, но при этом не отвлекать потенциальных пользователей на постоянные обсуждения, так как это, всё-таки, не основной их вид деятельности.

При анализе ожиданий пользователей используют различные формы взаимодействия. Так, например, интервью может использоваться для беседы с пользователем один на один. Часто применяется для обсуждения узкопрофильных требований, за работу с которыми отвечает конкретный уполномоченный представитель заказчика. Опросы или анкетирование применяются для того, чтобы выяснить некоторые нюансы требований (обсуждение различных вариантов) среди фокус-групп пользователей. Например, для выявления требования к информационной системе, отслеживающей успеваемость студентов некого учреждения высшего образования, нюансы требований для модулей, с которыми будут работать заведующие кафедрой, можно обсудить в форме опроса каждого заведующего, выяснив, какие нюансы есть в работе каждого из них. А вот нюансы требований для модуля, с которым будут работать непосредственно студенты, можно выяснить с помощью анкетирования

большей части студентов. Для решения более сложных вопросов, касающихся требований, можно проводить семинары (для больших групп пользователей) или мозговые штурмы (для небольших групп пользователей) совместно с бизнес-аналитиком, дизайнером или другими техническими специалистами со стороны разработчика. Активно к работе с требованиями могут привлекаться технические специалисты со стороны заказчика (если таковые имеются).

2.2. Анализ требований

При анализе собранных требований рассматриваются различные характеристики требований:

– завершённость (требование полностью, т. е. в полном объёме описано в одном месте);

– последовательность (непротиворечивость с другими требованиями);

– атомарность (требование не может быть разбито на более мелкие требования без потери завершённости);

– актуальность (требование не устарело с течением времени);

– выполнимость (требование может быть реализовано в пределах проекта, т. е. в пределах отведённого бюджета и временных рамок);

– недвусмысленность (формулировка требования не позволяет двоякого прочтения);

– верифицируемость (то есть выполнение требование может быть проверено каким-либо из способов, например, непосредственный осмотр работы готового функционала; демонстрация работы в специально созданных условиях; документально подтверждённые результаты тестирования с указанием процента функционала, покрытого тестированием, процента успешно пройденных критичных, средне-критичных и некритичных тестов; а также анализ технических характеристик программного продукта).

Для пояснения характеристик требований приведём некоторые примеры соблюдения и несоблюдения характеристик (см. таблицу 2.1).

Таблица 2.1 – Примеры требований

Характеристика	**Пример требования, не обладающего характеристикой**	**Пример требования, обладающего характеристикой**
Завершённость	*Требование #123* Список книг отображается в таблице с заголовками: – заглавие – автор – название издательства – год издания – количество страниц ***Прим**.:* *Поле «автор» – слишком размытое понятие, не хватает информации о том, что именно необходимо отобразить в этом столбце*	*Требование #123* Список книг отображается в таблице с заголовками: – заглавие – ***фамилия и инициалы автора*** – название издательства – год издания – количество страниц

Последовательность	*Требование #124* Таблица со списком книг по умолчанию должна быть отсортирована по столбцу «жанр» ***Прим.:*** *Это требование противоречит предыдущему требованию, в котором столбца «жанр» не было. Необходимо или исправить это требование, или предыдущее*	*Требование #123* Список книг отображается в таблице с заголовками: – ***название жанра*** – заглавие – фамилия и инициалы автора – название издательства – год издания – количество страниц
Атомарность	*Требование #123* Список книг отображается в таблице с заголовками: – название жанра – заглавие – фамилия и инициалы автора – название издательства – год издания – количество страниц ***Можно произвести сортировку списка по любому столбцу, кликнув по его заголовку***	*Требование #123* Список книг отображается в таблице с заголовками: – название жанра – заглавие – фамилия и инициалы автора – название издательства – год издания – количество страниц
		Требование #125 ***Можно произвести сортировку списка, формируемого в соответствии с требованием #123, по любому столбцу, кликнув по его заголовку***
Недвусмысленность	*Требование #234* На кнопке «Сохранить» должна отображаться красивая иконка ***Прим.:*** *Термин «красивая» субъективен и может пониматься разными людьми по-разному*	*Требование #234* На кнопке «Сохранить» должна отображаться ***иконка из файла "/img/save-icon.png"***
	Требование #235 Таблицы с большим количеством строк разбиваются на несколько страниц ***Прим.:*** *Неоднозначно, что такое «большое количество строк»*	*Требование #235* Таблицы с ***количеством строк более 50*** разбиваются на несколько страниц
	Требование #236 В случае длительного ожидания загрузки данных необходимо отобразить индикатор загрузки ***Прим.:*** *Неоднозначно, что такое «длительное ожидание»*	*Требование #236* В случае ожидания загрузки данных ***дольше 1,5 с*** необходимо отобразить индикатор загрузки

2.3. Документирование требований

Существуют различные уровни требований:

– бизнес-требование;

– пользовательские требования;

– функциональные требования;

– нефункциональные требования.

Бизнес-требование описывает общую цель создания информационной системы и глобальные задачи, которые она должна решать. Описывается в документе «Vision and Scope».

Пользовательские требования описывают те действия, которые могут сделать пользователи с различными ролями в системе. Пользовательские требования отвечают на вопрос «***что может сделать пользователь?***». Используются для описания общей концепции программного продукта. Также могут применяться как способ структуризации функциональных требований. Документируются с помощью UML-диаграмм вариантов использования (Use Case Diagram). Также такие диаграммы называются диаграммами прецедентов. Используются как в документе «Vision and Scope», так и в других документах. Помимо вариантов использования применяются ещё описание коротких действий, совершаемых пользователем в системе, которые называются User Stories.

Функциональные требования подробно описывают, каким образом должны функционировать те или иные части программного продукта. Именно в функциональных требованиях описываются все тонкости бизнес-логики приложения. Функциональные требования отвечают на вопрос «***как будет работать система?***». Основной вид требования, которым должны руководствоваться разработчики. Описываются в виде текста и могут сопровождаться различной графикой. В том числе для иллюстрации функциональных требований активно используется прототипирование. Описываются в документе «System Requirements Specification» или в специальном документе «Functional Requirements».

Нефункциональные требования описывают различные технические требования к информационной системе. Сложность выявления таких требований заключается в том, что заказчик, как правило, фокусируется лишь на функциональных требованиях, и может не представлять, какие нефункциональные требования критичны для программного продукта.

Примерами нефункциональных требований могут быть требования к дизайну и удобству использования, требования к безопасности и надёжности, требования к производительности и др.

Рассмотрим теперь некоторые дополнительные характеристики требований, важные для процесса планирования сроков и бюджета проекта.

– Важность (приоритетность). Характеристика показывает, насколько критичным для заказчика является выполнение этого требования.

Такая характеристика позволяет выбирать для реализации, прежде всего, самые критичные требования, что помогает концентрировать основные силы разработчиков на первоочередных задачах программного продукта.

– Сложность реализации (в человеко-часах). Характеристика позволяет планировать время работы над реализацией программного продукта. Также в сложности реализации может учитываться требуемая квалификация специалиста, который может реализовать требование. Часто сложность трудно оценить для всего требования, так как в будущем для реализации этого требования различным разработчикам будут ставиться различные задачи в рамках этого требования. В таком случае сложность оценивается для каждой задачи отдельно, с учётом квалификации разработчиков.

– Устойчивость (вероятность изменения в будущем). Характеристику трудно измерить некоторым конкретным числом. Как правило, используют одно из значений: высокая устойчивость (требование вряд ли изменится в будущем), средняя устойчивость (изменения возможны), низкая устойчивость (изменения очень вероятны).

2.4. Практическое задание

Описать требования для некоторого приложения (см. свой вариант), оформив их в виде набора Wiki-страниц на ресурсе https://gitlab.com/.

Для выполнения работы необходимо:

1) создать профиль на сайте GitLab;
2) создать новый проект (в левом верхнем углу сайта кнопка **Menu** > **Projects** > **Create New Project**);
3) перейти в раздел **Wiki** (см. список категорий слева);
4) для всех требований создать набор Wiki-страниц, имеющих ссылки друг на друга (см. требования ниже).

Самостоятельно выбрать наиболее подходящий с Вашей точки зрения тип приложения (Desktop-приложение, Web-приложение или Mobile-приложение).

Требования должны быть разбиты на уровни: бизнес-требование, пользовательские и функциональные. Также требования должны быть разбиты на категории. Уровни и категории должны выделяться заголовками и подзаголовками.

Формулировка требований должна соответствовать характеристикам:

– завершённость;
– последовательность;
– атомарность;
– выполнимость;
– недвусмысленность;
– верифицируемость.

Для каждого функционального требования должны быть указаны приоритет и устойчивость требования, а также описан способ верификации требования (осмотр, демонстрация, тестирование).

Стартовая Wiki-страница (страница Home) должна содержать название проекта в заголовке, бизнес-требование и список ссылок на страницы с пользовательскими требованиями. Список ссылок на пользовательские требования должен быть разбит на категории. Названия категорий оформить в виде подзаголовков.

Для каждого пользовательского требования необходимо создать отдельную Wiki-страницу, содержащую краткое название требования в заголовке, формулировку требования (что может сделать пользователь), список ролей пользователей, которым доступно данное действие, и список ссылок на функциональные требования, описанные в рамках данного пользовательского требования. Внизу страницы добавить ссылку «назад» на стартовую страницу.

Для каждого функционального требования необходимо создать отдельную Wiki-страницу, содержащую краткое название требования в заголовке, формулировку требования (как пользователь может осуществить действие, с подробным описанием нажимаемых кнопок и ссылок, заполняемых полей и т.д.). Внизу страницы добавить ссылку «назад» на страницу с родительским пользовательским требованием.

При редактировании Wiki-страницы снизу формы редактирования текста страницы есть ссылка на документацию с примерами оформления элементов Wiki-страницы. Также удобно пользоваться вкладкой **Preview** окна редактирования (если используется старая версия редактора).

Варианты задания:

1. ***Патентный отдел***

Приложение должно содержать сведения об авторах изобретений и научных экспертах, позволять авторам подавать заявки на изобретения, а экспертам оценивать заявки и принимать решение о выдаче авторских свидетельств.

2. ***Спортивные секции***

Приложение должно содержать сведения о спортивных секциях вуза, занимающихся в них студентах, расписании занятий секций, тренерах секции.

3. ***Издательство***

Приложение должно содержать сведения о книгах, авторах, сроках издания книги, расходных материалах на издание, оптовых покупателях изданий и о количестве проданных книг покупателями.

4. ***Банк***

Приложение должно позволять подавать клиентам заявки на открытие счетов, менеджерам банка - открывать счета, клиентам и кассирам - проводить кассовые операции (перевод денег с одного счёта на другой, пополнение счёта наличными, снятие наличными средств со счёта).

5. ***Телефонная компания***

Приложение должно содержать сведения о клиентах, предоставляемых им услугах, тарифных планах.

6. ***Туристическое агентство***

Приложение должно содержать сведения о туристических маршрутах, гостиницах, программе отдыха, заказах туров.

7. ***Библиотека***

Приложение должно содержать сведения о книгах, читателях, выдаче и возврате книг.

8. ***Аптека***

Приложение по управлению продажей лекарств в аптеке.

9. ***Подписка***

Приложение должно содержать сведения о периодических изданиях, подписчиках, стоимости подписки на различные издания и сроках подписки.

10. ***Конструкторское бюро***

Приложение должно содержать сведения о конструкторах, проектирующих оборудования (или здания) по заявкам заказчиков, технических заданиях на проекты и их статусах, а также о назначении конструкторов на проекты.

11. ***Ресторан***

Приложение должно содержать сведения о меню ресторана, разделённое на категории, стоимости блюд и должна позволять заказывать столик в ресторане с заказом блюд из меню.

12. ***Ломбард***

Приложение должно содержать сведения о сданных в ломбард вещах, статусе вещи (выкуплена, продана, и т.д.), стоимости залога, категориях вещей.

3. РАБОТА В КОНСОЛИ

Консоль – это практически единственный инструмент, позволяющий взаимодействовать с удалёнными серверами. Но и при работе с локальным персональным компьютером использование консоли позволяет ускорить и упростить ряд задач.

В этом разделе рассмотрим отдельные аспекты применения консоли для решения различных задач в операционных системах Windows и Linux.

3.1. Лабораторная работа: Основы работы в консоли Windows

Консоль Windows позволяет работать с операционной системой в режиме диалога. Пользователь вводит некоторую команду, операционная система выполняет эту команду и выводит результат работы команды на экран. После чего пользователь может вводить следующую команду.

В качестве команды может выступать одна из стандартных команд Windows или некоторая программа, название которой вводится с клавиатуры в командной строке.

Общий формат команды:

```
command [argument] [argument] [и т.д.]
```

`command` – это название команды или имя исполняемого файла (например, EXE-файла, BAT-файла, CMD-файла), при этом если указывается имя исполняемого файла, то его расширение может быть пропущено, например, вместо команды `abc.exe` можно вводить только команду `abc`.

`[argument]` –это параметр, передаваемый команде. Квадратные скобки при записи параметра не пишутся. Они указываются в справке по команде просто чтобы показать, что данный параметр является не обязательным, то есть команда может быть запущена без указания данного параметра. Список и правила указания параметров зависят от конкретной команды или исполняемого файла. Как правило в справке по самой команде или исполняемому файлу приводятся возможные параметры и правила их указания. Зачастую в консольных командах Windows справка по команде выводится с помощью параметра `/?`, например:

```
abc.exe /?
```

Но могут использоваться и другие способы (особенно для команд, портированных с операционной системы Unix). Например, `-h`, `--help` или просто запуск команды без параметров.

Список стандартных консольных команд Windows можно получить с помощью команды `help` без параметров.

Задание «Работа с файловой системой»

С помощью консольных команд выполнить следующие действия (используемые команды и результат каждого пункта показать преподавателю, рекомендуется делать скриншоты окна консоли и показывать всё задание в виде набора скриншотов):

1. С помощью команд `cd`, `dir`, `mkdir` создать в рабочем каталоге текущего пользователя каталог console и в нём структуру подкаталогов, показанную в столбце **А** таблицы с вариантами. Вывести в консоль дерево каталогов с помощью команды `tree`.

2. С помощью команды `copy` создайте в каталоге с названием, указанным в столбце **В**, файл с названием about.me, в который впишите свои фамилию, имя и номер группы на английском языке. С помощью команды `type` выведите содержимое этого файла в консоль.

Указание: для того, чтобы создать новый файл с помощью команды `copy`, необходимо в качестве имени файла-источника указать системное имя **`con`**. После ввода команды то, что будет набираться с клавиатуры, будет записываться в файл. Для того, чтобы завершить ввод данных в файл, необходимо нажать в консоли комбинацию клавиш Ctrl+Z, а затем Enter.

3. Выберите англоязычную песню, в которой есть не менее 3-х куплетов и припев, создайте в каталоге с названием, указанным в столбце **C** файлы с названием verse-1.txt, verse-2.txt и т.д., в каждый из которых впишите текст соответствующего куплета, а также создайте файл refrain (без расширения .txt), в который поместите строку "REFRAIN:", а далее — текст припева. В конце каждого файла добавьте по 2 пустые строки. Файлы можно создавать в блокноте Windowns (Notepad). С помощью команды `dir` выведите содержимое данного каталога, назовите преподавателю размер каждого файла.

4. Скопируйте в каталог с названием, указанным в столбце **D**, все файлы с расширением .txt из каталога, указанного в столбце **C**. Переименуйте с помощью команды `rename` файл refrain в каталоге, указанном в столбце **C**, в файл refrain.tmp. Выведите дерево всех подкаталогов и файлов в каталоге console с помощью команды `tree`.

5. С помощью команды `copy` объедините все файлы с расширением .txt из каталога, указанного в столбце **C**, в файл с именем song.out. Отобразите в консоли содержимое этого файла командой `type`, а затем выведите дерево каталогов командой `tree`.

6. Перенесите файл song.out в каталог vwxyz с помощью команды `move`, а всем файлам с расширением .txt в каталоге, указанном в столбце **C** установите атрибут "Скрытый" с помощью команды `attrib`. Выведите дерево каталогов. Выведите все файлы, в том числе и скрытые, из каталога, указанного в столбце **C**.

7. Измените с помощью команды `rename` расширение файла verse-2.txt на verse-2.tmp в каталоге, указанном в столбце **D**. Выведите в консоль все файлы с расширением .txt из каталога, указанного в столбце **D**. Удалите с помощью команды `del` все файлы с расширением .tmp из всех подкаталогов. Выведите дерево каталогов.

8. Перенесите с помощью команды `move` каталог vwxyz из каталога console в домашний каталог текущего пользователя, удалите с помощью команды `rmdir` каталог console со всем его содержимым. Выведите в консоль содержимое каталога vwxyz командой `dir` и содержимое файла song.out командой `type`.

Варианты

№	A	B	C	D
1	├──abc │ └──defgh ├──ijk │ └──lmnop ├──qrstu └──vwxyz	defgh	lmnop	qrstu
2	├──abcdef │ ├──ghijk │ └──lmnop ├──qrstu └──vwxyz	ghijk	lmnop	qrstu
3	├──abc │ └──defgh ├──ijklm ├──nop │ └──qrstu └──vwxyz	defgh	ijklm	qrstu
4	├──abcdef │ ├──ghijk │ ├──lmnop │ └──qrstu └──vwxyz	ghijk	lmnop	qrstu
5	├──abc │ └──defgh ├──ijklm ├──nopqr └──stu └──vwxyz	defgh	ijklm	nopqr
6	├──abcde ├──fghijk │ ├──lmnop │ └──qrstu └──vwxyz	abcde	lmnop	qrstu
7	├──abcde ├──fgh │ └──ijklm ├──nop │ └──qrstu └──vwxyz	abcde	ijklm	qrstu
8	├──abc │ ├──defgh │ └──ijklm └──nop ├──qrstu └──vwxyz	defgh	ijklm	qrstu

9	├──abcde ├──fgh │ └──ijklm ├──nopqr └──stu └──vwxyz	abcde	ijklm	nopqr
10	├──abcde ├──fghij └──klmnop ├──qrstu └──vwxyz	abcde	fghij	qrstu
11	├──abcde ├──fghij ├──klm │ └──nopqr └──stu └──vwxyz	abcde	fghij	nopqr
12	├──abcde └──fghijk ├──lmnop ├──qrstu └──vwxyz	abcde	lmnop	qrstu

Задание «Переменные окружения»

С помощью программы `series` (предоставляется преподавателем, её описание см. ниже) и с помощью перенаправления потоков разработайте командный файл, который находит минимальное, максимальное значение, сумму и среднее значение чисел из некоторого текстового файла, имя которого должно задаваться первым параметром командной строки вызова командного файла. Все результаты должны быть помещены в один текстовый файл, имя которого должно задаваться вторым параметром командной строки вызова командного файла.

Например, пусть есть некоторый файл a.txt со следующим содержимым:

```
5.6 9.0 3.4
7.8 1.2
7.6 3.2 0.1 5.4
9.8
```

Пусть разработанный командный файл имеет имя calculate.cmd, тогда запуск этого командного файла можно осуществить с помощью команды:

```
calculate.cmd a.txt b.txt
```

После запуска данной команды содержимое файла b.txt должно быть следующим (порядок строк может быть другим):

```
MIN: 0.1
MAX: 9.8
SUM: 53.1
AVG: 5.31
```

Указание:

Перебор значений агрегирующих функций (MIN, MAX, SUM, AVG) организовать с помощью команды `for`.

Самостоятельно изучить:

- обработку параметров, передаваемых командному файлу;
- перенаправление потоков ввода и вывода;
- команду `for` и способ выполнения нескольких команд в цикле `for`.

Описание программы series:

Программе на вход (поток STD_IN) подаётся некоторое количество чисел (если вызывать программу без перенаправления потоков, то эти значения через пробел, табуляцию, переход на новую строку, вводятся с клавиатуры, ввод заканчивается комбинацией клавиш Ctrl+Z). Данная программа считывает значение переменной окружения `FUNCTION`, которая задаёт имя некоторой агрегирующей функции. Возможные значения:

`MIN` – минимальное значение

`MAX` – максимальное значение

`SUM` – сумма значений

`AVG` – среднее арифметическое значений

При этом имя переменной окружения и возможные значения переменной окружения должны задаваться только большими буквами.

Результат выполнения программа выводит в поток STD_OUT.

Задание «Пакетная обработка файлов»

Создать командный файл, с помощью которого во всех текстовых файлах в текущем каталоге (файлы с расширением .txt) найти строки, содержащие указанную первым параметром командного файла подстроку (с помощью команды `find`). Затем занести все такие строки во временный файл out.tmp (если такой файл существует перед запуском командного файла, сразу завершить работу командного файла с соответствующей ошибкой, проверку и выход из командного файла осуществить командами `if` и `goto`, а вывод сообщения об ошибке командой `echo`). Вывести все строки из файла out.tmp в консоль командой `type`, отсортировав их командой **sort**. Объединение вывода содержимого файла командой `type` и сортировки командой `sort` можно осуществить с помощью специального символа «|». Общий синтаксис следующий:

```
commandA | commandB
```

В таком случае то, что выводит команда `commandA` в поток STD_OUT, направляется в поток STD_IN команды `commandB`.

После успешного вывода результата файл out.tmp удалить.

При штатной работе командного файла (если не было ошибки, связанной с уже имеющимся файлом out.tmp), командный файл должен выводить только результат поиска и сортировки. Сами выполняемые команды выводится не должны (для подавления вывода строки с самой командой перед

названием команды в командном файле ставится символ «@», например, `@del out.tmp`). Для отключения вывода в консоль результатов работы команды выполняется команда `echo off`, а для включения вывода обратно команда `echo on`.

Особенности выполнения задания смотри в вариантах, указанных в таблице ниже.

Варианты:

Вариант	**Команда `find`**		**Команда `sort`**
	Вывод номера строк	*Не учитывать регистр*	*Обратный порядок*
1	Да	Да	Нет
2	Нет	Да	Нет
3	Да	Нет	Нет
4	Нет	Нет	Нет
5	Да	Да	Да
6	Нет	Да	Да
7	Да	Нет	Да
8	Нет	Нет	Да

Задание «Перенаправление потоков ввода и вывода»

Ознакомьтесь с примером программы, обрабатывающей параметры командной строки:

```
#include <iostream>
#include <cstdlib>

int main(int argc, char* argv[])
{
    if (argc == 3)
    {
        int x = atoi(argv[1]);
        int y = atoi(argv[2]);
        std::cout << x << " + " << y << " = " << x + y << "\n";
    }
    else
    {
        std::cerr << "Program need exactly 2 parameters\n";
    }
    return 0;
}
```

Написать консольную программу на C++, которая находит решение одной из трёх задач (см. свой вариант). Тип задачи определяется с помощью значения единственного параметра командной строки — это один символ, показывающий тип решаемой задачи. Далее на стандартный поток вывода STD_OUT программа выводит сообщения, запрашивая у пользователя входные данные для задачи. Сами входные данные считываются со стандартного

потока ввода STD_IN. Если пользователь ввёл некорректные данные для задачи, то в стандартный поток вывода STD_OUT необходимо вывести только сообщение «Ошибка ввода входных данных», а подробное сообщение об ошибке необходимо выводить в стандартный поток ошибок (STD_ERR). При демонстрации работы программы преподавателю необходимо запустить программу в консоли для каждого из видов задач. Продемонстрировать как работу с корректными, так и работу с некорректными данными. Для каждой задачи и каждого входного набора данных осуществить запуск программы дважды:

1. Запуск осуществлять таким образом, чтобы потоки STD_IN и STD_OUT работали только с консолью, а поток STD_ERR перенаправлялся в файл с именем, указываемым при запуске (например, errors.txt). При этом каждый последующий запуск программы должен не перезаписывать этот файл, а лишь дописывать в его конец очередное сообщение об ошибке.

2. Запуск осуществлять таким образом, чтобы потоки STD_OUT и STD_IN были ассоциированы с файлами, имена которых должны указываться при запуске (например, in.txt и out.txt), а поток STD_ERR был связан с консолью.

Указание:

В коде самой программы работы с файлами быть не должно. Программа должна позволять вводить как целые, так и дробные числа. Обрабатывать ошибки ввода, если пользователь вводит нечисловое значение, не нужно. Однако, если пользователь вводит отрицательное число там, где оно должно быть положительным, программа должна выдавать ошибку. Общая последовательность и формат вывода сообщений должны быть такими же, как и в примерах ниже. Однако в примерах приведены лишь некоторые случаи ввода некорректных данных. Программа должна обрабатывать все возможные варианты ввода некорректных данных, кроме ввода нечисловых значений.

Виды решаемых задач:

А) Вычисление площади треугольника по длинам трёх сторон.

Пример вызова программы для корректных данных:

```
C:\Users\User> program A
************************************************************
* Вычисление площади треугольника по длинам трёх его сторон. *
************************************************************
Введите длину первой стороны:
3
Введите длину второй стороны:
5
Введите длину третьей стороны:
4
Длины сторон треугольника равны 3, 5 и 4
Площадь треугольника равна 6
```

Пример вызова программы для некорректных данных:

```
C:\Users\User> program A
************************************************************
* Вычисление площади треугольника по длинам трёх его сторон *
************************************************************
Введите длину первой стороны:
1.2
Введите длину второй стороны:
3.4
Введите длину третьей стороны:
5.6
Ошибка ввода входных данных
При вычислении площади треугольника по длинам трёх его сторон были
введены следующие значения длин сторон: 1.2, 3.4, 5.6.
Такой треугольник не может существовать, так как длина самой
длинной стороны должна быть меньше суммы длин двух других
его сторон
```

В) Длина хорды окружности, задаваемой координатами центра и радиусом, образуемой прямой, проходящей через заданные две точки.

Пример вызова программы для корректных данных:

```
C:\Users\User> program B
************************************************************
* Вычисление длины хорды окружности, лежащей на прямой, *
* проходящей через две заданные точки                    *
************************************************************
Введите координаты центра окружности:
3 5
Введите радиус окружности:
10
Введите координаты первой точки, задающей прямую, содержащую
хорду:
0 0
Введите координаты второй точки, задающей прямую, содержащую
хорду:
9 9
Центр окружности расположен в точке (3, 5); радиус окружности
равен 10
Хорда лежит на прямой, проходящей через точки (0, 0) и (9, 9)
Длина хорды равна 19.8
```

Пример вызова программы для некорректных данных:

```
C:\Users\User> program B
************************************************************
* Вычисление длины хорды окружности, лежащей на прямой, *
* проходящей через две заданные точки                    *
************************************************************
Введите координаты центра окружности:
0 0
```

```
Введите радиус окружности:
1
Введите координаты первой точки, задающей прямую, содержащую
хорду:
10 0
Введите координаты второй точки, задающей прямую, содержащую
хорду:
0 10
Ошибка ввода входных данных
Прямая, проходящая через точки (10, 0) и (0, 10), не пересекает
окружность с центром в точке (0, 0) и радиусом 1
```

C) Вычисление длины средней линии трапеции, заданной координатами четырёх вершин.

Пример вызова программы для корректных данных:

```
C:\Users\User> program C
****************************************
* Вычисление длины средней линии трапеции, *
* заданной координатами четырёх вершин     *
****************************************
Введите координаты первой вершины трапеции:
-2 -2
Введите координаты второй вершины трапеции:
-2 6
Введите координаты третьей вершины трапеции:
2 9
Введите координаты четвёртой вершины трапеции:
10 7
Трапеция с вершинами в точках (-2, -2); (-2, 6); (2, 9); (10, 7)
Длина средней линии равна 10
```

Пример вызова программы для некорректных данных:

```
C:\Users\User> program C
****************************************
* Вычисление длины средней линии трапеции, *
* заданной координатами четырёх вершин     *
****************************************
Введите координаты первой вершины трапеции:
0 0
Введите координаты второй вершины трапеции:
0 1
Введите координаты третьей вершины трапеции:
2 2
Введите координаты четвёртой вершины трапеции:
1 0
Ошибка ввода входных данных
Четырёхугольник с вершинами в точках
(0, 0); (0, 1); (2, 2); (1, 0) не является трапецией
```

D) Нахождение точки пересечения двух прямых, проходящих через две пары точек, заданных своими координатами.

Пример вызова программы для корректных данных:

```
C:\Users\User> program D
*****************************************************
* Нахождение точки пересечения двух прямых, заданных *
* двумя парами точек                                 *
*****************************************************
Введите координаты первой точки, задающей первую прямую:
-1 8
Введите координаты второй точки, задающей первую прямую:
1 5
Введите координаты первой точки, задающей вторую прямую:
5 3
Введите координаты второй точки, задающей вторую прямую:
7 4
Первая прямая проходит через точки (-1, 8) и (1, 5)
Вторая прямая проходит через точки (5, 3) и (7, 4)
Прямые пересекаются в точке (3, 2)
```

Пример вызова программы для некорректных данных:

```
C:\Users\User> program D
*****************************************************
* Нахождение точки пересечения двух прямых, заданных *
* двумя парами точек                                 *
*****************************************************
Введите координаты первой точки, задающей первую прямую:
0 0
Введите координаты второй точки, задающей первую прямую:
1 1
Введите координаты первой точки, задающей вторую прямую:
1.2 3.4
Введите координаты второй точки, задающей вторую прямую:
5.6 7.8
Ошибка ввода входных данных
Прямая, проходящая через точки (0, 0) и (1, 1), не пересекается
с прямой, проходящей через точки (1.2, 3.4) и (5.6, 7.8)
```

E) Определение координат четвёртой вершины ромба по заданным координатам первых трёх вершин (если таких вариантов несколько — рассмотреть их все.

Пример вызова программы для корректных данных:

```
C:\Users\User> program E
*****************************************************
* Определение координат неизвестной вершины ромба по *
* известным трём вершинам                            *
*****************************************************
```

```
Введите координаты первой вершины ромба:
2 1
Введите координаты второй вершины ромба:
6 4
Введите координаты третьей вершины ромба:
2 6
Если первые три вершины ромба находятся
в точках (2, 1); (6, 4) и (2, 6),
то четвёртая вершина находится в точке (6, 9)
```

Пример вызова программы для некорректных данных:

```
C:\Users\User> program E
**************************************************
* Определение координат неизвестной вершины ромба по *
* известным трём вершинам                          *
**************************************************
Введите координаты первой вершины ромба:
3 2
Введите координаты второй вершины ромба:
6 3
Введите координаты третьей вершины ромба:
5 5
Ошибка ввода входных данных
Заданы три точки с координатами (3, 2); (6, 3); (5, 5).
При этом расстояния между этими точками попарно являются
различными числами, а для того, чтобы эти точки задавали ромб,
необходимо, чтобы хотя бы одна пара расстояний была одинаковой,
так как длины сторон ромба должны быть равны
```

Варианты:

Вариант	**Задача**		
	1	**2**	**3**
1	A	B	C
2	A	B	D
3	A	B	E
4	A	C	D
5	A	C	E
6	A	D	E
7	B	C	D
8	B	C	E
9	B	D	E
10	C	D	E

3.2. Лабораторная работа: Основы работы в консоли Linux

Консоль Linux, так же, как и консоль Windows, позволяет работать с операционной системой в режиме диалога. Формат команды в Linux практически такой же, как и в Windows. Есть некоторые незначительные отличия, например, вместо символа «\» в Linux в качестве разделителя каталогов в пути использует символ «/».

Справку по команде и её параметрам в Linux можно получить с помощью команды `man`, первым параметром которой передаётся название команды. Например:

```
man cd
```

Задание «Работа с файловой системой»

С помощью консольных команд выполнить следующие действия (используемые команды и результат каждого пункта показать преподавателю, рекомендуется делать скриншоты окна консоли и показывать всё задание в виде набора скриншотов):

1. С помощью команд `cd`, `ls`, `mkdir` создать в рабочем каталоге текущего пользователя каталог console и в нём структуру подкаталогов, показанную в столбце **A** таблицы с вариантами. Вывести в консоль все каталоги рекурсивно с помощью команды `ls`.

2. С помощью команды редактора `nano` создайте в каталоге с названием, указанным в столбце **B**, файл с названием about.me, в который впишите свои фамилию, имя и номер группы на английском языке. С помощью команды `cat` выведите содержимое этого файла в консоль.

3. Выберите англоязычную песню, в которой есть не менее 3-х куплетов и припев, создайте в каталоге с названием, указанным в столбце **C** файлы с названием verse-1.txt, verse-2.txt и т.д., в каждый из которых впишите текст соответствующего куплета, а также создайте файл refrain (без расширения .txt), в который поместите строку "REFRAIN:", а далее – текст припева. В конце каждого файла добавьте по 2 пустые строки. С помощью команды `ls` выведите содержимое данного каталога, назовите преподавателю размер каждого файла.

4. Скопируйте в каталог с названием, указанным в столбце **D**, все файлы с расширением .txt из каталога, указанного в столбце **C**. Переименуйте с помощью команды `mv` файл refrain в каталоге, указанном в столбце **C**, в файл refrain.tmp. Выведите список всех подкаталогов и файлов рекурсивно в каталоге console с помощью команды `ls`.

5. С помощью команды `cp` объедините все файлы с расширением .txt из каталога, указанного в столбце **C**, в файл с именем song.out. Отобразите в консоли содержимое этого файла командой `cat`, а затем выведите список всех каталогов и файлов рекурсивно командой `ls`.

6. Перенесите файл song.out в каталог vwxyz с помощью команды `mv`, а затем скройте все файлы с расширением .txt в каталоге, указанном в столбце **C**, добавив символ «`.`» в начале названия файла. Выведите все каталоги и файлы рекурсивно (в том числе и скрытые).

7. Измените с помощью команды `mv` расширение файла verse-2.txt на verse-2.tmp в каталоге, указанном в столбце **D**. Выведите в консоль все файлы с расширением .txt из каталога, указанного в столбце **D**. Удалите с помощью команды `rm` все файлы с расширением .tmp из всех подкаталогов. Выведите все каталоги и файлы рекурсивно (в том числе и скрытые).

8. Перенесите с помощью команды `mv` каталог vwxyz из каталога console в домашний каталог текущего пользователя, удалите с помощью команды `rm` каталог console со всем его содержимым. Выведите в консоль содержимое каталога vwxyz командой `ls` и содержимое файла song.out командой `cat`.

Варианты:

Варианты возьмите такие же, что и в аналогичном задании по работе с файловой системой в консоли Windows.

Задание «Перенаправление потоков ввода и вывода»

Скомпилируйте и запустите программу, разработанную в рамках аналогичного задания по перенаправлению потоков ввода и вывода в консоли Windows. Требования к способам запуска и варианты задания взять такими же, что и в задании для Windows.

4. СИСТЕМА КОНТРОЛЯ ВЕРСИЙ GIT

В соответствии с заданием своего варианта (см. ниже) необходимо разработать консольное приложение по обработке некоторых данных, хранящихся в текстовых файлах. Шаги по реализации данного приложения необходимо сохранять в публичном репозитории одного из открытых ресурсов, поддерживающих систему контроля версий GIT (https://gitlab.com/, https://github.com/ и т.д.). Шаги должны быть аналогичны описанным в примере ниже. При обработке текстовых файлов считать, что: отдельные поля в строке разделяются ровно одним пробелом; вся информация в файле корректна; количество строк в файле не превосходит 100; длина одной строки в файле не превосходит 200 символов.

Пример

В качестве примера рассмотрим создание консольного приложения со следующими требованиями:

Вариант: 0
Название: *Библиотечный абонемент*
Формат входного файла
`фамилия_читателя имя_читателя отчество_читателя начало конец фамилия_автора имя_автора отчество_автора название`
`фамилия_читателя`, `имя_читателя`, `отчество_читателя` – строки, не содержащие пробелов – читатель библиотеки
`начало`, `конец` – даты, когда читатель взял книги в библиотеке и когда вернул в формате `дд.мм.гг`, где `дд`, `мм` и `гг` – это целые неотрицательные числа
`фамилия_автора`, `имя_автора`, `отчество_автора` – строки, не содержащие пробелов – автор книги
`название` – название книги – строка, которая может содержать в том числе и пробелы
Фильтрация данных
1. Вывести все записи на абонементе (читателя и книгу) с книгами Пушкина Александра Сергеевича.
2. Вывести все записи на абонементе (читателя и книгу), которые были взяты в библиотеке за март 2015-го года.

Шаги решения предложенной задачи

1. Скачать и установить консольный клиент GIT с официального сайта https://git-scm.com/. В консоли GIT сгенерировать SSH-ключ с помощью утилиты ssh-keygen:

```
ssh-keygen -t rsa -C "<e-mail>"
```

Здесь вместо `<e-mail>` указать свой почтовый адрес. При генерации ключа утилита запрашивает пароль, который нужно будет вводить каждый раз при выполнении операций с сервером (например, `git push`), но при этом пароль будет проверяться локально. Не требуется, чтобы этот пароль совпадал с паролем на GitLab или GitHub.

В своём профиле на GitLab открыть настройки (пункт меню «Preferences» под изображением пользователя в правом верхнем углу), выбрать подпункт «SSH Keys», скопировать в текстовое многострочное поле «Key» содержимое файла `.ssh/id_rsa.pub` из локальной папки текущего пользователя, ввести в текстовое однострочное поле «Title» описание ключа (например, «Ключ на моём ноутбуке»), нажать кнопку «Add key»

2. Создать на GitLab публичный пустой репозиторий и выполнить его клонирование с помощью консольной команды `git clone`.

Для нашего примера создадим репозиторий с названием `cpp-console-library-subscription` (`https://gitlab.com/yermochenko-example/cpp-console-library-subscription`).

Скопируем SSH-URL репозитория (кнопка «Clone», пункт «Clone with SSH»).

В консоли GIT создадим папку для GIT-репозиториев, например так:

```
cd d
mkdir git
cd git
```

Далее выполним клонирование репозитория и перейдём в папку с репозиторием:

```
git clone git@gitlab.com:yermochenko-example
                    /cpp-console-library-subscription.git
cd cpp-console-library-subscription
```

После перехода в папку с репозиторием убедитесь, что в консоли GIT после имени текущей папки, выделенной жёлтым цветом, указано название ветки (main) голубым цветом.

3. Создадим теперь проект в Visual Studio. Здесь рассматривается Visual Studio 2019 Community Edition. При открытии Visual Studio выберите пункт «Create a new project» и из списка шаблонов выберите язык программирования C++, платформу Windows, тип проекта Console, шаблон «Empty Project», нажмите кнопку «Next». На следующие форме убедитесь, что переключатель «Place solution and project in the same directory» выключен. Введите в поле «Location» папку с GIT-репозиториями (в нашем примере это `D:\git`). В поле «Solution» введите имя папки с созданным репозиторием (в нашем примере это `cpp-console-library-subscription`), если допустить ошибку в названии папки, то папка с репозиторием и папка с решением не будут никак связаны. В поле «Project name» введите имя проекта, например `main-project`. Нажмите кнопку «Create».

После создания проекта в Visual Studio в консоли GIT для проверки в папке репозитория выполним команду `git status`, после чего будет выведен список «Untracked files» (это список неотслеживаемых файлов), в примере это:

```
.vs/
cpp-console-library-subscription.sln
main-project/
```

4. Настройка репозитория: создание файла `.gitignore`

В консоли GIT в папке репозитория (в примере это папка `cpp-console-library-subscription`) выполним команду:

```
nano .gitignore
```

Откроется консольный текстовый редактор nano, в котором необходимо ввести следующий текст:

```
.vs/
Debug/
Release/
```

Далее с помощью комбинации клавиш Ctrl+O вызывается диалог сохранения файла. Имя файла не меняем, нажимаем клавишу Enter. После этого будет создан файл с указанным именем. Выход из текстового редактора осуществляется клавишей Ctrl+X.

Если теперь выполнить команду `git status`, можно увидеть, что в списке «Untracked files» отсутствует папка `.vs/` (так как она теперь с списке игнорируемых), но добавился файл `.gitignore`. Выполним первый Commit:

```
git add .gitignore
git commit -m "Инициализация репозитория"
git push
```

Команда `git add` добавляет файл из списка «Untracked files» в список «Staged files» (это список файлов, изменения в которых будут зафиксированы). Команда `git commit` выполняет фиксацию изменений в локальном репозитории. Команда `git push` сохраняет изменения, которые были зафиксированы в локальном репозитории, в удалённый репозиторий.

После выполненной последовательности команд можно в браузере обновить страницу с репозиторием. Добавленный файл должен появиться в репозитории на GitLab. Также в консоли GIT можно выполнить команду `git log`, которая выводит информацию о последних выполненных операциях.

5. Сделаем теперь изменения в проекте в Visual Studio и сохраним эти изменения. Добавим в проект `main-project` в папку с исходными файлами (Source Files) файл `main.cpp` со следующим содержимым:

```
#include <iostream>

int main()
{
    std::cout << "Hello\n";
    return 0;
}
```

После этого в консоли GIT выполним последовательность команд:

```
git add --all
git status
```

Таким образом, как видно, команда `git add` позволяет добавлять в список «Staged files» все новые и изменившиеся файлы автоматически без необходимости их перечисления. Выполним теперь фиксацию изменений и синхронизацию с удалённым репозиторием:

```
git commit -m "Создана заготовка проекта"
git push
```

6. На этом шаге внесём изменения в проект через браузер сразу в удалённом репозитории. Для этого на сайте GitLab обновите страницу с проектом, откройте папку с проектом, щёлкнув на имени папки `main-project` (столбец «Name»), но не на Commit message «Создана заготовка проекта» (столбец «Lasy commit»). Далее аналогично откройте файл `main.cpp`, щёлкнув на его имени. На открывшейся странице нажмите кнопку «Edit» и в открывшемся редакторе измените текст программы на следующий:

```
#include <iostream>

using namespace std;

int main()
{
```

```
    cout << "Laboratory work #8. GIT\n";
    cout << "Variant #0. Library Subscription\n";
    cout << "Author: Sergey Yermochenko\n";
    return 0;
}
```

При выполнении задания своего варианта нужно вписать номер и название своего варианта, а также свои имя и фамилию.

После чего в поле «Commit message» вписать сообщение «Добавлен вывод общей информации о задании» и нажать внизу кнопку «Commit changes».

В консоли GIT выполнить команду:

```
git pull
```

После чего убедиться, что внесённые изменения появились в проекте в Visual Studio. Скомпилировать и запустить проект и убедиться в его работоспособности.

На данном шаге мы потренировались забирать в локальный репозиторий изменения, внесённые кем бы то ни было в удалённый репозиторий. Однако практика редактирования кода прямо на сайте GitLab является далеко не самой хорошей, так как мы не имеем возможности перед фиксацией изменений (операцией Commit) убедиться в работоспособности исходного кода. Фиксировать изменения в репозитории (локальном или удалённом) можно только с работоспособным кодом. Допускаются случаи, когда неработоспособный код (возможно, даже не компилирующийся) фиксируется в репозитории в специальный ветках, не мешающих основному коду.

7. На следующем шаге создадим отдельную ветку для создания файла с данными. Для этого в консоли GIT выполним команду:

```
git checkout -b data
```

Обратите внимание, что после имени текущей папки ветка `main` (голубым шрифтом) изменилась на `data`.

После создания ветки в Visual Studio добавим в проект `main-project` текстовый файл `data.txt` (в контекстном меню папки проекта выберем пункт Add > New Item, в открывшемся диалоговом окне выберем категорию Visual C++ > Utility, тип файла Text File (.txt) и введём имя файла `data.txt`), затем поместим в него информацию, согласно требований варианта, в нашем примере это:

```
Иванов Иван Иванович 01.03.2015 22.03.2015 Пушкин Александр
                                          Сергеевич Евгений Онегин
Петров Пётр Петрович 01.03.2015 11.03.2015 Лермонтов Михаил
                                                  Юрьевич Мцыри
Сидоров Сидор Сидорович 06.03.2015 30.03.2015 Достоевский Фёдор
                                         Михайлович Братья Карамазовы
Петров Пётр Петрович 11.03.2015 23.03.2015 Пушкин Александр
                                           Сергеевич Капитанская дочка
Петров Пётр Петрович 23.03.2015 25.04.2015 Лев Николаевич
                                                Толстой Война и мир
```

```
Иванов Иван Иванович 24.03.2015 13.04.2015 Достоевский Фёдор
                                          Михайлович Братья Карамазовы
Иванов Иван Иванович 16.04.2015 11.05.2015 Лев Николаевич
                                               Толстой Анна Каренина
Петров Пётр Петрович 25.04.2015 04.05.2015 Пушкин Александр
                                             Сергеевич Евгений Онегин
Петров Пётр Петрович 04.05.2015 15.05.2015 Достоевский Фёдор
                                          Михайлович Братья Карамазовы
```

Выполним в консоли GIT операции Add и Commit с сообщением «Добавлены начальные данные». После этого выполним операцию Push. Но, так как в удалённом репозитории такой ветки ещё нет, то команда должна иметь следующий вид:

```
git push --set-upstream origin data
```

После чего можно убедиться, что ветка и изменения в ней появились в удалённом репозитории.

8. На этом шаге выполним слияние ветки `main` с веткой `data`:

```
git checkout main
git merge data
git push
```

Первая команда – переключение на другую ветку. Переключаемся в ту ветку, в которую хотим влить изменения из другой ветки.

Вторая команда – вливаем в текущую ветку изменения из указанной ветки.

Третья команда – синхронизируем изменения с удалённым репозиторием.

9. На этом шаге необходимо создать отдельную ветку `dev`. В этой ветке добавить в проект заголовочный файл для описания структуры данных, в нашем примере это будет файл `book_subscription.h`, а также специальный заголовочный файл для констант `constants.h`.

Файл `constants.h`:

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

#define MAX_FILE_ROWS_COUNT 100
#define MAX_STRING_SIZE 200

#endif
```

Файл `book_subscription.h`:

```
#ifndef BOOK_SUBSCRIPTION_H
#define BOOK_SUBSCRIPTION_H

#include "constants.h"

struct date
{
    int day;
    int month;
```

```
    int year;
};

struct person
{
    char first_name[MAX_STRING_SIZE];
    char middle_name[MAX_STRING_SIZE];
    char last_name[MAX_STRING_SIZE];
};

struct book_subscription
{
    person reader;
    date start;
    date finish;
    person author;
    char title[MAX_STRING_SIZE];
};

#endif
```

Теперь необходимо выполнить операции Add, Commit с сообщением «Добавлены структуры для хранения данных», затем Push для новой ветки dev, Megre в ветку main и Push ветки main.

10. Как правило, выполнение фиксации изменений в ветке dev в больших проектах выполняют отдельные разработчики, а вот слияние ветки dev с другими ветками выполняет другой (старший) разработчик. В нашем случае ветка dev сливается с веткой main, но в реальных проектах система веток обычно более сложная. Рассмотрим вариант, когда выполняется несколько фиксаций изменений в разных ветках, которые потом объединяются в одну ветку.

Переключимся в ветку dev и добавим два файла следующего содержания:

Файл file_reader.h:

```
#ifndef FILE_READER_H
#define FILE_READER_H

#include "book_subscription.h"

void read(const char* file_name, book_subscription* array[],
                                                    int& size);

#endif
```

Файл file_reader.cpp:

```
#include "file_reader.h"
#include "constants.h"

#include <fstream>
#include <cstring>
```

```
date convert(char* str)
{
    date result;
    char* context = NULL;
    char* str_number = strtok_s(str, ".", &context);
    result.day = atoi(str_number);
    str_number = strtok_s(NULL, ".", &context);
    result.month = atoi(str_number);
    str_number = strtok_s(NULL, ".", &context);
    result.year = atoi(str_number);
    return result;
}

void read(const char* file_name, book_subscription* array[],
                                                    int& size)
{
    std::ifstream file(file_name);
    if (file.is_open())
    {
        size = 0;
        char tmp_buffer[MAX_STRING_SIZE];
        while (!file.eof())
        {
            book_subscription* item = new book_subscription;
            file >> item->reader.last_name;
            file >> item->reader.first_name;
            file >> item->reader.middle_name;
            file >> tmp_buffer;
            item->start = convert(tmp_buffer);
            file >> tmp_buffer;
            item->finish = convert(tmp_buffer);
            file >> item->author.last_name;
            file >> item->author.first_name;
            file >> item->author.middle_name;
            file.read(tmp_buffer, 1); // чтение лишнего пробела
            file.getline(item->title, MAX_STRING_SIZE);
            array[size++] = item;
        }
        file.close();
    }
    else
    {
        throw "Ошибка открытия файла";
    }
}
```

Модифицируем также файл `main.cpp` следующим образом:

```
#include <iostream>

using namespace std;
```

```
#include "book_subscription.h"
#include "file_reader.h"
#include "constants.h"

int main()
{
   setlocale(LC_ALL, "Russian");
   cout << "Лабораторная работа №8. GIT\n";
   cout << "Вариант №0. Библиотечный абонемент\n";
   cout << "Автор: Сергей Ермоченко\n\n";
   book_subscription* subscriptions[MAX_FILE_ROWS_COUNT];
   int size;
   try
   {
      read("data.txt", subscriptions, size);
      for (int i = 0; i < size; i++)
      {
         cout << subscriptions[i]->reader.last_name << '\n';
         cout << subscriptions[i]->reader.first_name << '\n';
         cout << subscriptions[i]->reader.middle_name << '\n';
         cout << subscriptions[i]->finish.day << ' ';
         cout << subscriptions[i]->finish.month << ' ';
         cout << subscriptions[i]->finish.year << '\n';
         cout << subscriptions[i]->start.day << ' ';
         cout << subscriptions[i]->start.month << ' ';
         cout << subscriptions[i]->start.year << '\n';
         cout << subscriptions[i]->author.last_name << '\n';
         cout << subscriptions[i]->author.first_name << '\n';
         cout << subscriptions[i]->author.middle_name << '\n';
         cout << subscriptions[i]->title << '\n';
         cout << '\n';
      }
      for (int i = 0; i < size; i++)
      {
         delete subscriptions[i];
      }
   }
   catch (const char* error)
   {
      cout << error << '\n';
   }
   return 0;
}
```

Выполним в ветке dev фиксацию этих изменений с сообщением «Реализовано форматирование данных при выводе в консоль» и операцию Push. **Важно**: выполняем операции только в ветке dev без выполнения её слияния в ветку main.

После этого переключимся в ветку `data`, в которой модифицируем файл с данными, добавив в него несколько строк:

Файл `data.txt`:

```
Иванов Иван Иванович 01.03.2015 22.03.2015 Пушкин Александр
                                          Сергеевич Евгений Онегин
Петров Пётр Петрович 01.03.2015 11.03.2015 Лермонтов Михаил
                                                  Юрьевич Мцыри
Сидоров Сидор Сидорович 06.03.2015 30.03.2015 Достоевский Фёдор
                                        Михайлович Братья Карамазовы
Васильев Василий Васильевич 09.03.2015 09.04.2015 Лев
                                     Николаевич Толстой Война и мир
Петров Пётр Петрович 11.03.2015 23.03.2015 Пушкин Александр
                                       Сергеевич Капитанская дочка
Петров Пётр Петрович 23.03.2015 25.04.2015 Лев Николаевич
                                             Толстой Война и мир
Иванов Иван Иванович 24.03.2015 13.04.2015 Достоевский Фёдор
                                        Михайлович Братья Карамазовы
Васильев Василий Васильевич 10.04.2015 03.05.2015 Пушкин
                               Александр Сергеевич Капитанская дочка
Иванов Иван Иванович 16.04.2015 11.05.2015 Лев Николаевич
                                           Толстой Анна Каренина
Петров Пётр Петрович 25.04.2015 04.05.2015 Пушкин Александр
                                          Сергеевич Евгений Онегин
Петров Пётр Петрович 04.05.2015 15.05.2015 Достоевский Фёдор
                                        Михайлович Братья Карамазовы
Васильев Василий Васильевич 05.05.2015 02.06.2015 Лев
                                     Николаевич Толстой Анна Каренина
Васильев Василий Васильевич 03.06.2015 29.06.2015 Достоевский
                                   Фёдор Михайлович Братья Карамазовы
```

А также внесём изменения в ту версию файла `main.cpp`, которая есть в ветке `data`, добавив в него вывод номера студенческой группы, в которой учится автор программы:

Файл `main.cpp`:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Laboratory work #8. GIT\n";
    cout << "Variant #0. Library Subscription\n";
    cout << "Author: Sergey Yermochenko\n";
    cout << "Group: XX\n";
    return 0;
}
```

Выполним теперь операции Commit с сообщением «Добавлены тестовые данные» и Push в ветку `data`. **Важно**: выполняем операции только в ветке `data` без выполнения её слияния в ветку `main`.

11. На этом этапе посмотрим, как выполняется разрешение конфликтов при слиянии веток. Сначала переключимся в ветку `main`, затем выполним слияние в неё ветки `dev`, а затем слияние в неё ветки `data`. На последней операции слияния `git merge` получим ошибку:

```
Auto-merging main-project/main.cpp
CONFLICT (content): Merge conflict in main-project/main.cpp
Automatic merge failed; fix conflicts and then commit the result.
```

Откроем теперь в Visual Stuio файл `main.cpp`, содержащий конфликты. Из этого файла нужно будет вручную удалить строки:

```
<<<<<<< HEAD
=======
>>>>>>> data
```

Привести код в необходимое корректное состояние (оставить вывод группы на русском языке в начале программы сразу после вывода имени и фамилии автора). После чего выполнить операцию Commit с сообщением «Разрешён конфликт слияния» и операцию Push.

12. На этом шаге необходимо переключиться в ветку `dev`, выполнить слияние в неё ветки `main`. Далее реализовать функцию для фильтрации данных со следующим прототипом:

```
<data_structure>** <function_name> (
  <data_structure>* array[],
  int size,
  bool (*check)(<data_structure>* element),
  int& result_size
);

/*
ОПИСАНИЕ ФУНКЦИИ <function_name>:
  функция перебирает массив с исходными данными и все указатели
  на элементы, для которых функция отбора возвращает значение
  true, помещаются в новый массив, указатель на который
  возвращается функцией

ПАРАМЕТРЫ:
  array       - массив с исходными данными
  size        - размер массива с исходными данными
  check       - указатель на функцию отбора.
                В качестве значения этого параметра можно
                передать имя функции отбора, прототип которой
                приведён ниже
  result_data - параметр, передаваемый по ссылке - переменная, в
                которую функция запишет размер результирующего
                массива

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ
  указатель на массив из указателей на элементы, удовлетворяющие
  условию отбора (для которых функция отбора возвращает true)
*/
```

```
bool <check_function_name> (<data_structure>* element);

/*
ОПИСАНИЕ ФУНКЦИИ <check_function_name>:
  функция отбора - проверяет, удовлетворяет ли один элемент
                   условию отбора

ПАРАМЕТРЫ:
  element - указатель на элемент, который нужно проверить

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ
  true, если элемент удовлетворяет условию отбора, и false в ином
  случае
*/
```

А также в соответствии с вариантом реализовать две функции отбора. Выбор критерия отбора должен осуществляться пользователем. Пример реализации смотри в репозитории `cpp-console-library-subscription`. Выполнить Commit и Push в ветку `dev`, а затем Merge в ветку `main`.

13. На этом шаге полностью самостоятельно необходимо переключиться в ветку `dev`. Далее реализовать две функции сортировки данных с одинаковым прототипом, каждая из которых должна реализовать один из методов сортировки (см. вариант задания). Каждая из функций сортировки должна принимать в качестве параметров массив указателей на структуру с данными, размер массива и указатель на функцию сравнения.

Необходимо реализовать две функции сравнения, реализующие различные критерии сравнения элементов массива (см. вариант задания). Функции сравнения должны принимать в качестве параметров два указателя на структуру с данными, далее функция должна сравнивать два экземпляра структуры в соответствии с требуемым критерием сравнения, а затем возвращать целое число, равное нулю, если два экземпляра равны, или число, меньшее нуля, если первый параметр меньше второго, или число более нуля, если первый параметр больше второго.

Выбор метода сортировки и критерия сравнения должен осуществляться пользователем. Для этого в меню необходимо добавить ещё один пункт. При выборе этого пункта пользователь должен выбирать сначала один из двух методов сортировки. Затем выбирать один из двух критериев сортировки. При этом необходимо сначала получить указатель на функцию сортировки, соответствующую выбору пользователя. Затем получить указатель на функцию сравнения, соответствующую выбору пользователя. А затем через первый указатель вызвать функцию, передав ей в качестве параметра второй указатель.

Для удобства получения указателей на функции их можно поместить в два массива и осуществлять выбор по индексу.

Выполнить Commit и Push в ветку `dev`, а затем Merge в ветку `main`.

Варианты:

Вариант: 1

Название: *Результаты марафона*

Формат входного файла

`номер фамилия имя отчество старт финиш клуб`

`номер` – натуральное число – порядковый номер участника марафона

`фамилия`, `имя`, `отчество` – строки, не содержащие пробелов – участник марафона

`старт`, `финиш` – время в формате `чч:мм:сс`, где `чч`, `мм` и `сс` – это целые неотрицательные числа

`клуб` – название спортивного клуба, за который выступает участник – строка, которая может содержать в том числе и пробелы

Фильтрация данных

1. Вывести всех участников и их результаты из клуба «Спартак».
2. Вывести всех участников, у которых результат лучше чем 2:50:00.

Сортировка данных

Методы сортировки:

1. Сортировка обменом (Bubble sort)
2. Быстрая сортировка (Quick sort)

Критерии сортировки:

1. По возрастанию времени забега (разницы между временем финиша и старта)
2. По возрастанию названия спортивного клуба, а в рамках одного клуба по возрастанию фамилии участника

Вариант: 2

Название: *Программа конференции*

Формат входного файла

`начало конец фамилия имя отчество тема`

`начало`, `конец` – время начала и окончания доклада в формате `чч:мм`, где `чч` и `мм` – это целые неотрицательные числа

`фамилия`, `имя`, `отчество` – строки, не содержащие пробелов – автор доклада

`тема` – тема доклада – строка, которая может содержать в том числе и пробелы

Фильтрация данных

1. Вывести все доклады Иванова Ивана Ивановича.
2. Вывести все доклады длительностью больше 15 минут.

Сортировка данных

Методы сортировки:

1. Пирамидальная сортировка (Heap sort)
2. Сортировка слиянием (Merge sort)

Критерии сортировки:

1. По убыванию длительности доклада (разницы между временем окончания и начала доклада)
2. По возрастанию фамилии автора доклада, а в рамках одного автора по возрастанию темы доклада

Вариант: 3

Название: *Осадки*

Формат входного файла

`день месяц количество характеристика`

`день`, `месяц` – целые неотрицательные числа

`количество` – дробное неотрицательное число – объём выпавших в этот день осадков

`характеристика` – вид осадков (дождь, снег, мокрый снег и т.д.) – строка, которая может содержать в том числе и пробелы

Фильтрация данных

1. Найти все дни, в которые шёл дождь.

2. Найти все дни, в которые объём осадков был меньше 1,5.

Сортировка данных

Методы сортировки:

1. Шейкерная сортировки (Shaker sort)
2. Быстрая сортировка (Quick sort)

Критерии сортировки:

1. По возрастанию количества осадков
2. По возрастанию характеристики, а в рамках одной характеристики по возрастанию номера месяца, а в рамках одного месяца по возрастанию номера дня

Вариант: 4

Название: *Курсы валюты*

Формат входного файла

`банк покупка продажа адрес`

`банк` – строка, не содержащая пробелов – название банка

`покупка`, `продажа` – дробное неотрицательное число – стоимость некоторой иностранной валюты в местной валюте

`адрес` – адрес отделения банка – строка, которая может содержать в том числе и пробелы

Фильтрация данных

1. Вывести курсы валюты во всех отделениях банка «Беларусбанк» (с адресами).
2. Вывести курсы валюты и адреса отделений банков, в которых продажа меньше 2,5.

Сортировка данных

Методы сортировки:

1. Сортировка обменом (Bubble sort)
2. Сортировка слиянием (Merge sort)

Критерии сортировки:

1. По убыванию разницы между стоимостью продажи и покупки
2. По возрастанию названию банка, а в рамках одного банка по возрастанию адреса отделения

Вариант: 5

Название: *Протокол работы в Интернет*

Формат входного файла

`начало конец получено отправлено программа`

`начало`, `конец` – время начала и окончания использования сети Интернет в формате `чч:мм:сс`, где `чч`, `мм` и `сс` – это целые неотрицательные числа

`получено`, `отправлено` – количество байт, полученных и отправленных за время данного сеанса связи – это целые неотрицательные числа

`программа` – полный путь к программе, которая осуществляла подключение к сети Интернет – строка, которая может содержать в том числе и пробелы

Фильтрация данных

1. Вывести протокол использования сети Интернет программой Skype.
2. Вывести протокол использования сети Интернет после 8:00:00.

Сортировка данных

Методы сортировки:

1. Сортировка вставками (Insertion sort)
2. Быстрая сортировка (Quick sort)

Критерии сортировки:

1. По убыванию времени использования сети Интернет (разницы между временем конца и начала использования)
2. По возрастанию названия программы, а в рамках одной программы по убыванию суммарного количества переданных и полученных байт

Вариант: 6

Название: *Роза ветров*

Формат входного файла

`день месяц направление скорость`

`день`, `месяц` – целые неотрицательные числа

`направление` – строка, не содержащая пробелов – направление ветра, которое может быть одним из следующих значений: North, NorthEast, East, SouthEast, South, SouthWest, West, NorthWest

`скорость` – дробное неотрицательное число – скорость ветра в м/с

Фильтрация данных

1. Вывести все дни, в которые дул ветер в одном из направлений West, NorthWest или North.
2. Вывести все дни, в которые дул ветер больше 5 м/с.

Сортировка данных

Методы сортировки:

1. Шейкерная сортировки (Shaker sort)
2. Сортировка слиянием (Merge sort)

Критерии сортировки:

1. По убыванию скорости ветра
2. По возрастанию направления ветра, а в рамках одного направления по возрастанию номера месяца, а в рамках одного месяца по возрастанию номера дня

Вариант: 7

Название: *Банковские операции*

Формат входного файла

`дата время вид счёт сумма назначение`

`дата` – дата совершения операции в формате `дд.мм.гг`, где `дд`, `мм` и `гг` – это целые неотрицательные числа

`время` – время совершения операции в формате `чч:мм:сс`, где `чч`, `мм` и `сс` – это целые неотрицательные числа

`вид` – строка, не содержащая пробелов – вид операции, который может быть одним из следующих значений: приход, расход

`счёт` – строка, не содержащая пробелов – номер счёта, на который переводились средства (для расходной операции), или номер счёта, с которого переводились средства (для приходной операции)

`сумма` – дробное неотрицательное число – сумма операции

`назначение` – назначение платежа – строка, которая может содержать в том числе и пробелы

Фильтрация данных

1. Вывести все банковские приходные операции.
2. Вывести все банковские операции за ноябрь 2021 года.

Сортировка данных

Методы сортировки:

1. Сортировка выбором (Selection sort)
2. Быстрая сортировка (Quick sort)

Критерии сортировки:

1. По возрастанию назначения платежа
2. По возрастанию номера счёта, а в рамках одного счёта по возрастанию вида операции, а в рамках одного вида операции по убыванию суммы операции

Вариант: 8
Название: *Итоги сессии*
Формат входного файла
`фамилия имя отчество дата оценка дисциплина`
`фамилия`, `имя`, `отчество` – строки, не содержащие пробелов – студент
`дата` – дата сдачи экзамена в формате `дд.мм.гг`, где `дд`, `мм` и `гг` – это целые неотрицательные числа
`оценка` – это целое неотрицательное число в диапазоне между 1 или 10
`дисциплина` – название дисциплины – строка, которая может содержать в том числе и пробелы
Фильтрация данных
1. Вывести всех студентов и их оценки по дисциплине «История Беларуси».
2. Вывести всех студентов и дисциплины с оценками выше 7 баллов.

Сортировка данных
Методы сортировки:
1. Сортировка вставками (Insertion sort)
2. Сортировка слиянием (Merge sort)

Критерии сортировки:
1. По возрастанию фамилии студента
2. По возрастанию названия дисциплины, а в рамках одной дисциплины по убыванию оценки

Вариант: 9
Название: *Телефонные разговоры*
Формат входного файла
`номер дата время продолжительность тариф стоимость`
`номер` – строка, не содержащая пробелов – номер вызываемого абонента
`дата` – дата совершения звонка в формате `дд.мм.гг`, где `дд`, `мм` и `гг` – это целые неотрицательные числа
`время` – время начала звонка в формате `чч:мм:сс`, где `чч`, `мм` и `сс` – это целые неотрицательные числа
продолжительность – продолжительность звонка в формате `чч:мм:сс`, где `чч`, `мм` и `сс` – это целые неотрицательные числа
`тариф` – строка, не содержащая пробелов – название тарифа, которое может быть одним из следующих значений: город, межгород, международный, мобильный
`стоимость` – дробное неотрицательное число – стоимость одной минуты разговора по данному тарифу
Фильтрация данных
1. Вывести все телефонные разговоры на мобильные телефоны.
2. Вывести все телефонные разговоры в ноябре 2021 года.

Сортировка данных
Методы сортировки:
1. Пирамидальная сортировка (Heap sort)
2. Быстрая сортировка (Quick sort)

Критерии сортировки:
1. По убыванию продолжительности разговора
2. По возрастанию номера телефона, а в рамках одного номера по убыванию стоимости разговора

Вариант: 10
Название: *Каталог товаров*
Формат входного файла
`стоимость количество категория название`
`стоимость` – дробное неотрицательное число – стоимость одной единицы товара

количество – это целое неотрицательное число – количество единиц товара на складе
категория – категория товара – строка, не содержащая пробелов
название – название товара – строка, которая может содержать в том числе и пробелы

Фильтрация данных

1. Вывести все товары в категории «Промтовары».
2. Вывести все товары стоимостью выше 100 рублей.

Сортировка данных

Методы сортировки:

1. Сортировка выбором (Selection sort)
2. Сортировка слиянием (Merge sort)

Критерии сортировки:

1. По убыванию количества товара на складе
2. По возрастанию категории товара, а в рамках одной категории по убыванию стоимости

СПИСОК РЕКОМЕНДОВАННЫХ ИСТОЧНИКОВ

1. Страуструп, Б. Язык программирования C++. Краткий курс, 2-е издание. – Санкт-Петербург: Диалектика, 2019. – 320 с.

2. Вигерс, К.И. Разработка требований к программному обеспечению. – Москва: Русская Редакция, 2004. – 576 с.

3. Chacon, S. Straub, B. Pro Git [Электронный ресурс]. – Режим доступа: https://git-scm.com/book/ru/v2. – Дата доступа: 10.03.2022.

Учебное издание

ЕРМОЧЕНКО Сергей Александрович
ПЕТРОВА Татьяна Константиновна

ИНСТРУМЕНТАРИЙ РАЗРАБОТЧИКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические рекомендации

Технический редактор *Г.В. Разбоева*
Компьютерный дизайн *В.Л. Пугач*

Подписано в печать .2022. Формат 60x84 $^{1}/_{16}$. Бумага офсетная.
Усл. печ. л. 3,02. Уч.-изд. л. 2,55. Тираж экз. Заказ .

Издатель и полиграфическое исполнение – учреждение образования
«Витебский государственный университет имени П.М. Машерова».

Свидетельство о государственной регистрации в качестве издателя,
изготовителя, распространителя печатных изданий
№ 1/255 от 31.03.2014.

Отпечатано на ризографе учреждения образования
«Витебский государственный университет имени П.М. Машерова».
210038, г. Витебск, Московский проспект, 33.