МИНОБРНАУКИ РФ

ФГБОУ Тверской государственный технический университет

Кафедра «Программного Обеспечения»

Курсовая Работа

по дисциплине «Основы программирования»

на тему: «Модульное программирование в процедурах и функциях»

Работу выполнили: студенты группы

Б.ПИН.РИС-22.06

Иванов Алексей, Соболев Роман, Аванесян Григорий,

Борздов Марк, Мельников Михаил, Запецкий Вадим

Работу проверил: преподаватель

Биллиг Владимир Арнольдович

Тверь 2023

**Оглавление**

[**Глава 1. Вводная часть** 2](#_Toc133581844)

[**1.1.** **Декомпозиция – основная парадигма современного программирования.** 2](#_Toc133581845)

[**1.2. Структура Решения, содержащая три проекта как пример декомпозиции на верхнем уровне.** 3](#_Toc133581846)

[**1.3. Модульное программирование. Модуль, как совокупность сервисов, дающих решение задачи.** 4](#_Toc133581847)

[**1.4. Процедуры и функции.** 4](#_Toc133581848)

[**1.5. Корректность** 6](#_Toc133581849)

[**1.6. Тестирование** 8](#_Toc133581850)

[**1.7. Особенности построения интерфейса, ориентированного на пользователя.** 10](#_Toc133581851)

[**Глава 2. Практическая часть** 11](#_Toc133581852)

[**2.1. Пример интерфейсного проекта** 11](#_Toc133581853)

[**2.2. Пример реализации проекта библиотеки классов** 13](#_Toc133581854)

[**2.3. Пример тестирующего проекта** 14](#_Toc133581855)

[**2.4. Пример цельного проекта** 15](#_Toc133581856)

[**2.5. Общий список выполненных нами работ** 18](#_Toc133581857)

[**Глава 3. Заключение** 20](#_Toc133581858)

[**Список литературы.** 22](#_Toc133581859)

[**Приложения.** 23](#_Toc133581860)

[**Приложение 1. Мнения участников команды о командной работе** 23](#_Toc133581861)

[**Приложение 2. «О распределении обязанностей при написании курсовой»** 25](#_Toc133581862)

# **Глава 1. Вводная часть**

Разработка программного обеспечения – трудное занятие, требующее научного инженерного подхода для достижения наибольшей эффективности при решении задач, возникающих в ходе работы.

* 1. **Декомпозиция – основная парадигма современного программирования.**

Один из главных приемов программирования, позволяющий справиться со сложной задачей состоит в декомпозиции задачи – сведении сложной задачи к совокупности более простых, которые могут быть выполнены отдельно друг от друга, а затем объединены, для того чтобы сформировать общее решение.

Сложный проект должен быть декомпозирован на классы, а класс в свою очередь должен быть декомпозирован так, чтобы все его методы были простыми процедурами и функциями.

Говоря о декомпозиции, предполагается, что речь идет о декомпозиции программы – алгоритма, дающего решение задачи. Структура данных может быть настолько же сложна, как и алгоритм их обработки. Следовательно декомпозиция этих данных также важна, как и декомпозиция самого алгоритма.

**1.2. Структура Решения, содержащая три проекта как пример декомпозиции на верхнем уровне.**

Раскрывая тему декомпозиции, следует начать с декомпозиции проекта на верхнем уровне. Разбивая проект, как сложную задачу на несколько простых, мы получаем структурированное решение, которое содержит три подпроекта – библиотеку классов, интерфейсный проект и разработанную систему тестов.

Библиотека классов – это проект, который содержит множество классов, функций и методов, которые, как правило, содержат в себе алгоритмы для решения конкретных задач.

Интерфейсный проект отвечает за взаимодействие проекта с пользователем, предоставление ему данных и их обработку, как следствие и результата работы приложения.

Система тестов позволяет проверить корректность решений каждого подпроекта, метода или функции. Она помогает выявить ошибки в коде, что значительно может ускорить разработку всего проекта.

Таким образом, структура решения, содержащая три проекта, позволяет декомпозировать программный проект на менее сложные и управляемые компоненты. Каждый проект отвечает за конкретную функциональность программного решения, что делает его более удобным для разработки.

**1.3. Модульное программирование. Модуль, как совокупность сервисов, дающих решение задачи.**

Модульное программирование – это такой подход к написанию программного кода, где код делится на отдельные части – модули.

Модуль содержит данные и функции их обработки. Другим модулям нежелательно иметь собственные средства обработки этих данных, они должны пользоваться для этого функциями первого модуля.

Разделение программы на максимально обособленные части является сложной задачей, которая должна решаться на этапе проектирования.

Во время работы программы могут возникать ситуации, выходящие за пределы, предусмотренные спецификацией. Такие ситуации называются исключительными. В таком случае следует упомянуть об **устойчивости** –способности программной системы реагировать на исключительные ситуации. Обработка исключительных ситуаций – это процесс, направленный на достижение устойчивости.

Устойчивость системы понимается и в более широком смысле. Программная система устойчива, если при малых изменениях в спецификации достаточно внести малые изменения в коде системы, возвращающие систему в корректное состояние. Декомпозиция программного кода на достаточно простые модули способствует как достижению корректности, так и достижению устойчивости системы.

## **1.4. Процедуры и функции.**

Процедуры и функции - важные понятия в программировании, которые позволяют создавать модульные и масштабируемые программы. Они могут быть реализованы в виде сервисов - независимых компонентов программы, предоставляющих определенный функционал и выполняющих определенные задачи.

**Декомпозиция на уровне модуля** — это процесс разделения сложных программных модулей на более мелкие и простые компоненты. Это позволяет упростить сложные задачи на более маленькие подзадачи, что делает программу более понятной, легко поддерживаемой и тестируемой.

**Сведение к "простым" процедурам и функциям** — это подход, при котором сложные алгоритмы и логика программы разбиваются на более простые процедуры или функции. Это позволяет снизить сложность кода и сделать его более модульным, что упрощает его разработку, отладку и поддержку.

**Важность процедур, функций и декомпозиции на уровне модуля заключается в следующих аспектах:**

**1.** Модульность: процедуры и функции позволяют создавать модульный код, который легко понимать и поддерживать. Каждая процедура или функция выполняет конкретную задачу, что упрощает понимание и использование кода.

**2.** Повторное использование кода: когда логика программы разбита на маленькие процедуры или функции, это позволяет повторно использовать эти компоненты в различных частях программы или даже в различных проектах, что экономит время и ресурсы на разработке.

**3.** Упрощение разработки и отладки: разделение сложных задач на более маленькие и простые подзадачи делает разработку и отладку кода более простой и эффективной. Ошибки могут быть обнаружены и исправлены легче, а также тестирование кода становится более простым и надежным.

**4.** Улучшение поддержки и расширяемости: когда код разбит на модули, это позволяет вносить изменения и добавлять новые функции с минимальным воздействием на другие части программы. Это делает программу более гибкой и легко поддерживаемой в долгосрочной перспективе, а также облегчает ее расширение новым функционалом.

**5.** Улучшение понимания и сотрудничества: процедуры, функции и декомпозиция на уровне модуля позволяют разработчикам лучше понимать и взаимодействовать с кодом друг друга. Модульный подход упрощает командную работу, так как разработчики могут независимо работать над различными модулями программы, что улучшает разделение труда и повышает эффективность работы.

**6.** Улучшение производительности: оптимизация отдельных процедур и функций может быть более эффективной, чем оптимизация всей программы в целом. Более маленькие и простые процедуры или функции могут быть оптимизированы отдельно, что может привести к повышению производительности программы в целом.

В целом, использование процедур, функций и декомпозиции на уровне модуля является важным аспектом разработки программного обеспечения, который позволяет создавать модульный, масштабируемый, поддерживаемый и эффективный код. Это способствует повышению качества программного обеспечения, снижению сложности кода и улучшению сотрудничества между разработчиками, что в итоге может сэкономить время, ресурсы и улучшить производительность разработки программ.

**1.5. Корректность**

В контексте проектного управления означает соответствие проекта определенным требованиям, нормам, стандартам, правилам или ожиданиям, установленным заказчиком, заинтересованными сторонами или внутренними стандартами организации. Корректность также может быть связана с точностью, надежностью, безопасностью и соответствием проектных решений и результатов заданным целям и ожиданиям.

**Корректность является одним из основных критериев успешного проекта и важным аспектом проектного управления по нескольким причинам:**

**1.** Удовлетворение требований заказчика: корректность позволяет проекту соответствовать требованиям заказчика, что является ключевым фактором успешного завершения проекта и удовлетворения ожиданий заказчика. Если проект не соответствует требованиям заказчика, это может привести к недовольству заказчика, возникновению споров и дополнительным затратам на исправление ошибок. Однако также необходимо упомянуть то, что в таком случае не менее важно получить однозначные и понятные спецификации от заказчика, иначе будет невозможно корректно выполнить заказ.

**2.** Управление рисками: корректность является важным аспектом управления рисками. Некорректные решения, ошибки или недостатки в проекте могут привести к возникновению рисков, таких как отклонения от плана, потери бюджета, потери репутации и другие нежелательные последствия.

**3.** Качество результата: корректность также влияет на качество результата проекта. Корректные решения и реализация проекта согласно определенным стандартам и правилам обеспечивают высокое качество результата проекта, что способствует удовлетворению заинтересованных сторон и достижению поставленных целей.

**4.** Эффективность ыпроекта: корректность также может влиять на эффективность проекта. Корректное выполнение проектных решений и соответствие проекта установленным требованиям и стандартам может уменьшить количество ошибок, переработок и временных и финансовых затрат, что способствует более эффективному выполнению проекта.

**5.** Удовлетворение заинтересованных сторон: корректность так же важна для удовлетворения ожиданий других заинтересованных сторон, таких как участники проекта, руководство организации, партнеры проекта и другие. Корректность проекта может способствовать созданию доверия и хороших отношений с заинтересованными сторонами, что в свою очередь может повысить поддержку проекта и обеспечить успешное его выполнение.

Таким образом, корректность является важным критерием построения проекта, поскольку она обеспечивает соответствие проекта требованиям, стандартам и ожиданиям заказчика и других заинтересованных сторон, влияет на качество результата, управление рисками, эффективность проекта, а также удовлетворение ожиданий заинтересованных сторон. Внимание к корректности проекта помогает снижать риски и обеспечивать успешное завершение проекта с высокими показателями качества и удовлетворенности заинтересованных сторон.

## **1.6. Тестирование**

Тестирование — это процесс проверки программного продукта или проекта на соответствие определенным требованиям и оценки его качества. Оно является одним из важных этапов разработки программного обеспечения и выполняется с целью выявления ошибок, дефектов и недостатков, которые могут влиять на работоспособность, безопасность и удовлетворение потребностей пользователей.

Виды тестирования:

1. Юнит-тестирование: тестирование отдельных компонентов программы, таких как функции, классы или модули, для проверки их правильной работы в изоляции от остальной системы. Юнит-тесты позволяют рано выявлять дефекты и обеспечивать стабильность и надежность кода.

2. Нагрузочное тестирование выполняется с целью оценки производительности системы и ее способности справиться с определенной нагрузкой или объемом работы. Оно может включать проверку реакции системы на высокую нагрузку, обработку большого количества данных или одновременное использование системы множеством пользователей.

3. Тестирование пользователей, также известное как тестирование на применимость, выполняется для оценки удовлетворенности пользователями продуктом и его соответствия их ожиданиям и потребностям. Тестирование пользователей может включать сбор обратной связи от реальных пользователей, проведение опросов, интервью, анализ поведения пользователей и других методов для оценки опыта пользователей.

4. Функциональное тестирование проводится с целью проверки соответствия функциональных требований продукта. Оно включает в себя проверку того, выполняет ли продукт свои предполагаемые функции и работает ли он корректно согласно ожиданиям и требованиям пользователей. Функциональное тестирование может включать тестирование интерфейса пользователя, ввода/вывода данных, обработки ошибок и других функциональных аспектов продукта.

**Тестирование имеет высокую значимость для проекта:**

**1.** Обеспечение качества: оно позволяет выявлять ошибки и дефекты в проекте, что помогает обеспечить высокое качество программного продукта. Ошибки, найденные на ранних стадиях разработки, могут быть исправлены до того, как они окажутся в продукте, что помогает избежать проблем в дальнейшем.

**2.** Снижение рисков: оно дает возможность обнаружить потенциальные риски и проблемы, связанные с функциональностью, производительностью, безопасностью и другими аспектами проекта. Это позволяет предпринять меры по устранению этих рисков и снизить возможные негативные последствия.

**3.** Улучшение пользовательского опыта: оно позволяет проверить проект на соответствие ожиданиям и потребностям пользователей. Это позволяет выявлять и исправлять ошибки, которые могут отрицательно сказаться на удовлетворенности пользователей и улучшать пользовательский опыт.

**4.** Экономия ресурсов: выявление и исправление ошибок на ранних этапах разработки проекта позволяет избежать дополнительных затрат на исправление проблем в более поздние стадии, когда они могут стать более сложными и затратными.

**5.** Увеличение надежности: тестирование позволяет установить степень надежности проекта и его компонентов, что способствует повышению доверия к продукту со стороны пользователей, заказчиков и других заинтересованных сторон.

Таким образом, тестирование является важным инструментом для обеспечения корректности и качества проекта, снижения рисков, повышения надежности и удовлетворенности пользователей, оптимизации процесса разработки и повышения конкурентоспособности продукта на рынке.

**1.7. Особенности построения интерфейса, ориентированного на пользователя.**

Интерфейсный проект отвечает за взаимодействие программы с пользователем, соответственно есть несколько общих принципов, на которых основывается построение самого интерфейса.

Проектирование пользовательского интерфейса должно быть таким, чтобы в нём разобрался даже ребёнок. Хороший интерфейс — тот, который понятен и удобен пользователю, чтобы он сразу нашёл нужную кнопку и не блуждал в поисках по всей странице. Это — главная задача в построении интерфейса.

Также, как наша команда выяснила на личном опыте, нередко в проект интерфейса необходимо добавлять справку, в которой должно находиться объяснение каждого элемента управления интерфейса.

Задача проектировщика интерфейса — объединить компоненты пользовательского интерфейса в единую систему так, чтобы они были связаны стилем, композицией и задачей. Тогда пользователю сразу будет ясно, куда двигаться и в какой последовательности переходить от одного элемента к другому.

Помимо основных принципов построения, стоит отметить, что элементы интерфейса должны быть четко видны пользователю, сам интерфейс должен быть приятен глазу, то есть нужно тщательно подходить к выбору стиля, шрифта и размера элементов. Также интерфейсный проект должен содержать определенный набор инструментов для дополнительного взаимодействия с пользователем, дабы, например, при неверном вводе данных - оповестить пользователя об этом.

# **Глава 2. Практическая часть**

Для демонстрации вышеизложенных принципов инженерного подхода к разработке ПО можно использовать выполненные нашей командой по ходу семестра проекты, так как они разрабатывались на основе структурного подхода и в них его принципы соблюдаются.

Общая линия прослеживается во всех проектах, поэтому каждый разбирать подробно не имеет смысла, исходя из этого, приведем примеры подпроектов, полученных декомпозицией исходной задачи на 3 части.

## **2.1. Пример интерфейсного проекта**

Хороший интерфейс должен быть простым и интуитивно понятным для пользователя. Это означает, что он должен быть спроектирован таким образом, чтобы пользователю было легко найти и использовать все необходимые функции.

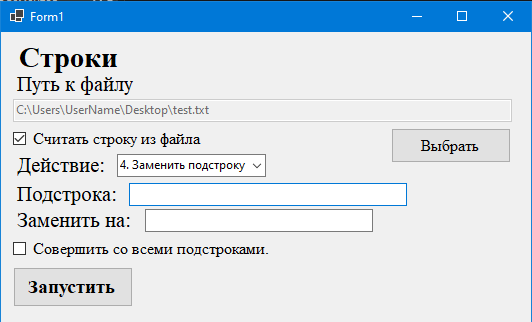
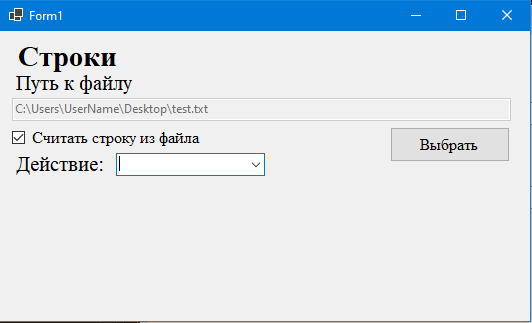
Эстетический интерфейс должен быть приятен глазу и не вызывать раздражения.

В сумме, чем более легким в использовании, привлекательным и функциональным является интерфейс, тем эффективнее его использование пользователем.

В наших работах эти условия выполняются.

На примере проекта **String**, позволяющего пользователю осуществлять различные действия над строками:

Интерфейс сделан простым и минималистичным, чтобы пользователи могли легко находить то, что им нужно. В проекте используется четкий шрифт и наиболее контрастные цвета. Таким образом, пользовательский интерфейс легко читаемый и приятен глазу. Сам интерфейс исключает возможность возникновения некоторых ошибок. Элементы формы становятся доступными для заполнения только тогда, когда это требуется.



Обработка данных из формы происходят в одном классе – entryPoint, где вызываются методы в зависимости от того, какое действие было выбрано. Такая реализация позволяет легко поддерживать и расширять функционал программы.

В целом, реализация интерфейсного модуля является корректной, эстетически верной и функционально правильной, потому что уделяется внимание пользовательскому опыту, соблюдаются стандарты дизайна, учитываются функциональные требования и используются лучшие практики программирования.

## **2.2. Пример реализации проекта библиотеки классов**

Библиотека классов проекта «Converse» - одна из наиболее ярких примеров правильной декомпозиции на верхнем уровне. Задачу перевода чисел из одной системы счисления в другую мы разбили на два этапа:

1) Перевод числа из системы счисления, заданной пользователем в десятичную.

2) Перевод числа из десятичной системы счисления в систему, также заданную пользователем.

Код декомпозиции на верхнем уровне выглядит следующим образом:

static public string ConvertSysms(string num, int baseOfNum, int resBase)

{

string tempRes = ConvertToDec(num, baseOfNum);

string res = ConvertFromDec(tempRes, resBase);

return res;

}

После же мы решили декомпозировать каждую из функций перевода на отдельный перевод дробной части и перевод целой части. Эту задачу мы реализовали в функциях **ConvertToDec** и **ConvertFromDec**. Принципы структурного программирования помогли сделать наш код более понятным для чтения, а метод проектирования «сверху-вниз» помог правильно декомпозировать поставленную задачу.

Для каждой из функций в проекте “Converse” написаны соответствующие заголовочные комментарии, помогающие понять, как работает та или иная функция нашего проекта

Пример нашего заголовочного комментария для функции **ConvertSysms**:

/// <summary>

/// Представляет метод, конвертирующий заданное число из системы счисления p

/// в систему счисления q, где p, q находятся в диапозоне [2; 36],

/// число представляет строку, содержащую целую и дробную часть,

/// разделенную точкой либо запятой.

/// </summary>

/// <param name="num">Конвертируемое число в формате строки</param>

/// <param name="baseOfNum">Система счисления конвертируемого числа</param>

/// <param name="resBase">Желаемая система счисления</param>

/// <returns>Число в необходимой системе счисления в формате строки</returns>

/// </summary>

Декомпозиция проекта «Converse» - одна из наиболее удачных декомпозиций наших проектов. Без правильного разделения основной задачи на подзадачи, эффективность итогового проекта падает, а построение алгоритма становится в разы сложнее.

## **2.3. Пример тестирующего проекта**

Существует множество причин, по которым важно проводить unit-тестирование. Прежде всего, это позволяет выявлять ошибки в коде на ранних стадиях разработки, что значительно сокращает время и стоимость исправления этих ошибок. Тестирование также помогает подтвердить, что изменения в коде не приводят к неожиданным результатам в других частях программы.

Unit-тестирование также упрощает процесс разработки, поскольку тесты можно автоматизировать. Это позволяет проводить тестирование быстрее и более эффективно, что в свою очередь позволяет разработчикам быстрее выявлять и исправлять ошибки.

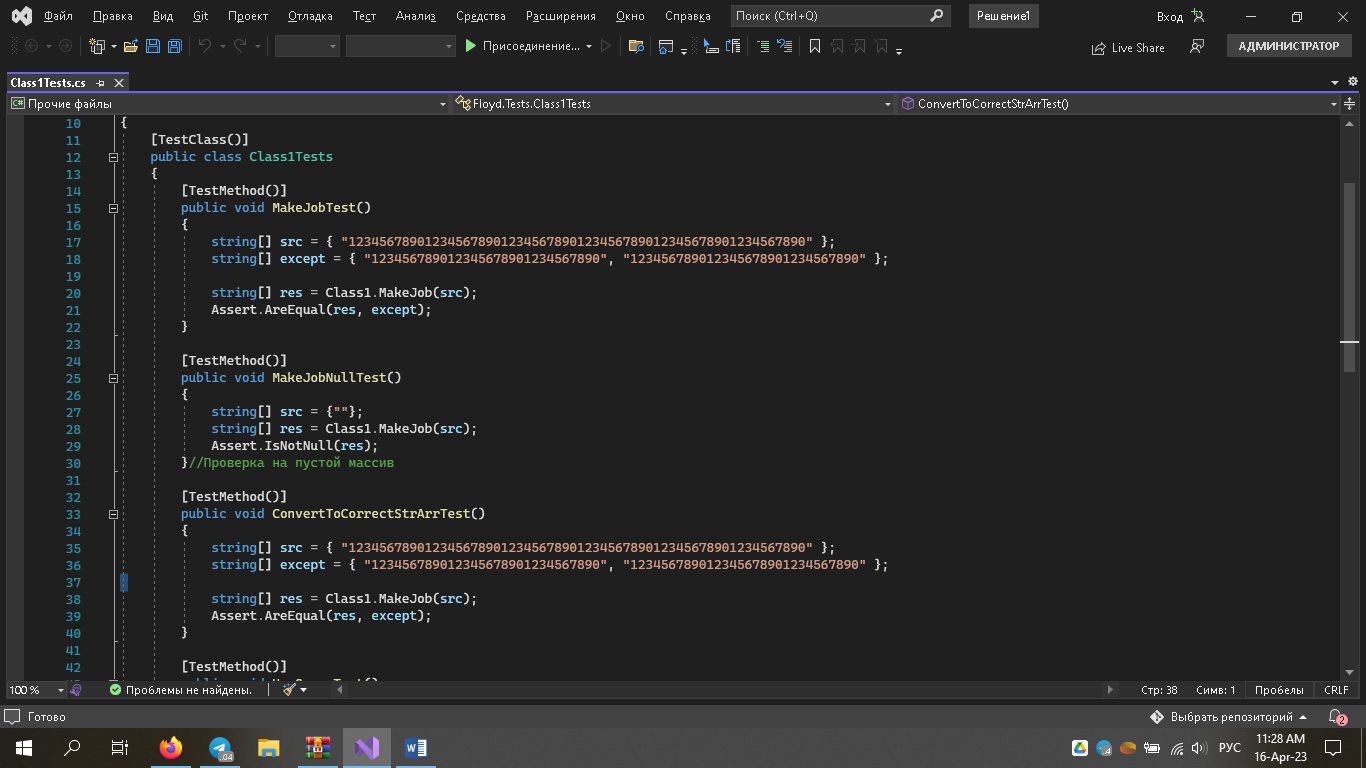
Наконец, проведение unit-тестирования способствует улучшению качества кода в целом. Это происходит благодаря тому, что разработчик должен написать тесты для каждой функции или метода, что заставляет его думать о том, каким должен быть код, чтобы он выполнял требуемые функции.

В целом, проведение unit-тестирования является неотъемлемой частью разработки ПО и позволяет значительно улучшить качество и надежность программного обеспечения.

Продемонстрировать Unit – тестирование можно на примере проекта Floyd, то есть нашей контрольной работы. Здесь наша команда тестировала каждую функцию проекта, что позволило нам находить ошибки гораздо быстрее.

Важный навык в тестировании — это креативность. Тестировщику нужно придумать такие ситуации, которые потенциально могут возникнуть. Безусловно, предусмотреть все возможные случаи неординарного поведения программы, невозможно, однако, чем нестандартнее подход, тем лучше.

То, что представлено на скриншоте, лишь малая часть проделанной работы по тестированию. В нашем проекте, мы старались проработать как можно больше ситуаций. Были протестированы функции ввода, обработки, и вывода информации. Мы старались принять сторону неподготовленного пользователя программы, намеренно вводя некорректные значения, не имеющие смысла, чтобы заставить наш проект работать неправильно.



## **2.4. Пример цельного проекта**

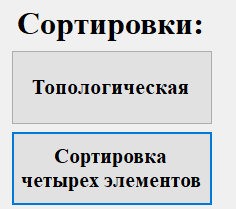
Показать, как между собой связываются вышеописанные подпроекты можно на основе проекта «TopSort» - наиболее сложного из всех проектов, которые нам приходилось реализовывать. Реализация функций требовала понимания многих новых для нас терминов и алгоритмов.

**Интерфейс.** Интерфейс нашего проекта разбит на две основные части:

1) Топологическая сортировка

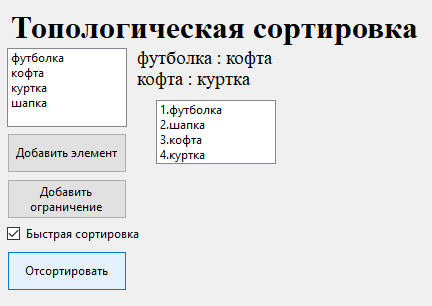
2) Сортировка четырех элементов

Выглядит это следующим образом:



При нажатии на кнопку "Топологическая" открывается новое окно, в котором представлен полный функционал сортировки. В этом окошке находятся несколько кнопок, при нажатии на которые осуществляются различные действия. Кнопка "Добавить элемент" отвечает за добавление элементов в сортировку. Кнопка "Добавить ограничение" позволяет пользователю добавить зависимость одного элемента от другого. Включенный флажок около надписи "Быстрая сортировка" осуществляет топологическую сортировку за квазилинейное время.

Если же пользователь выбирает "Сортировка четырех элементов", то открывается окно, в котором пользователь может ввести в строку 4 типа элементов с номером каждого из них и осуществить сортировку.



Результат сортировки

Зависимости

**Основной класс проекта.** Работу по реализации функций мы разделили на две разные программы. Первая из них это "Топологическая сортировка», вторая "Сортировка четырех элементов". Для каждой функции обеих программ у нас написан заголовочный, позволяющий просто и быстро ориентироваться по имеющемуся коду.

1) Верхний уровень нашей топологической сортировки разбит на два подкласса: "Быстрая топологическая сортировка", "Топологическая сортировка". Декомпозиция "Топологической сортировки" выглядит следующим образом:

public List<T> TopSort(List<T> elements, List<Pair<T>> constrains)

{

List<T> sorted = new();

T item;

while (elements.Count != 0)

{

item = FindItem(constrains, elements); //поиск минимального элемента

sorted.Add(item);// добавление элемента в итоговый список

RemoveFromElements(elements, item); //удаление текущего минимального элемента

RemoveFromConstrains(constrains, item); //удаление списков ограничений содержащий минимальный элемент

}

return sorted;

}

## **2.5. Общий список выполненных нами работ**

**Проект Forest:**

Задача данного проекта состоит в том, чтобы найти такой ряд с деревьями, в котором имеется достаточно места для посадки новых. Причем слева и справа в том же ряду тоже должны расти деревья. Работая над данным проектом, мы впервые столкнулись с работой в команде. Однако, грамотное распределение обязанностей в команде позволило нам быстро реализовать решение данной задачи.

**Проект Converse:**

Задача данного проекта заключается в переводе числа из системы счисления p в систему счисления q, где p и q находятся в диапазоне [2;36].

Именно в этом проекте мы впервые поняли, что для взаимодействия с пользователем необходимо добавить такой элемент управления, как «Справка».

Нашей командой было успешно реализованы библиотека классов, тестирование, а также интерфейсная часть проекта.

**Проект Sorts:**

Задача проекта Sorts состояла в сортировке элементов разных типов. При реализации библиотеки классов для данного проекта, мы открыли для себя достаточно много новых и эффективных алгоритмов сортировки, также успешно реализовали интерфейс и тестирование.

**Проект Floyd:**

Данный проект – это задача Роберта Флойда. Она заключается в том, чтобы найти в файле строку, состоящую только из пробелов и вернуть все предыдущие строки текста, устранив лишние пробелы между словами.

Наша команда, даже столкнувшись с трудностями, смогла решить данную задачу, реализовав декомпозицию на высоком уровне, построив библиотеку классов, а также разработав интерфейсный проект.

**Проект String**:

Проект String, исходя из его названия предполагает работу со строковыми структурами данных. Работая над этим проектом, наша команда успешно реализовала методы, обрабатывающие эти данные, в частности были реализованы такие функции, как нахождение подстроки в строке, удаление её, замена на другую подстроку и другие не менее важные методы обработки строки. Но самым главным являлся алгоритм Кнута–Морриса–Пратта, с его реализацией наша команда справилась успешно, а также были реализованы тестирующие и интерфейсные проекты.

**Проект TopSort:**

И наконец, последний наш проект **TopSort**, задача которого состоит в том, чтобы для заданного на конечном множестве ациклического отношения **r** найти отношение полного порядка, для которого **r** является подмножеством.

Особенность алгоритма заключается в том, что прежде, чем сортировать элементы, за линейное время строятся специальные структуры данных.

Благодаря прочитанной нам лекции наша команда успешно справилась с задачей реализации быстрой топологической сортировки. Была построена библиотека классов, интерфейсный проект, а также разработано тестирование.

# **Глава 3. Заключение**

В заключении работы следует рассказать о еще одном важном следствии декомпозиции – возможности работать в команде, а также об итогах семестровой работы.

Профессиональные программисты в подавляющем большинстве случаев работают не в одиночку, распределяя обязанности между собой, тем самым увеличивая эффективность работы. В самом деле, когда исходный проект четко разделен на подпроекты, а они в свою очередь на отдельные, независимые друг от друга функции, куда проще организовать командную деятельности. Легко заметить, что построение совместной работы таким образом идеально сочетается с парадигмой структурного программирования.

Правильная декомпозиция и разделение обязанностей, помимо всего прочего, улучшают микроклимат в команде, что также положительно влияет на эффективность работы.

И тем не менее, даже идеально зная теорию структурного программирования, зная о важности разбиения на модули, отдельные процедуры, функции, не всегда получается сразу действовать в соответствии со знаниями на практике. Связано это может быть с разными факторами, но главный из них – неопытность. Проектируя последние проекты, мы уже практически механически решали, каким образом его декомпозировать. При том, что сложность проектов лишь росла, время их выполнения даже немного падало. Все это показывает рост навыков каждого из участников команды в программировании в целом и в структурном программировании в частности. Более того, его принципы мы стали применять и вне дисциплины «Основы программирования», так как модульное программирование в процедурах и функциях доказало свою эффективность и его принципы мы усвоили.

Закончить данную работу хочется, подведя общефилософские итоги. Командная работа, наличие четких сроков выполнения и распределение обязанностей приближают нас, еще студентов, к профессиональной, «взрослой» деятельности. В таких условиях не только оттачиваются навыки написания кода, изучаются новые возможности языка, не только усваивается научный подход к разработке ПО, но и формируются ответственность, чувство времени и лидерские качества. Программирование играет важную роль в нашей жизни, но помимо него в ней есть еще очень много разных аспектов. Прекрасно и одновременно удивительно, что, казалось бы, такое далекое от обычной жизни понятие, как «структурное программирование», помимо всего прочего, еще и положительно влияет на личностные качества человека.

# **Список литературы.**

1. C/C++. Программирование на языке высокого уровня / Т. А. Павловская. — СПб.: Питер, 2003. —461 с: ил.
2. Лекция с YouTube-канала «Vladimir Billig» «Декомпозиция данных. Теорема SVD. Приложения»
3. https://studfile.net/preview/4494392/
4. https://studfile.net/preview/7679991/page:85/
5. https://studfile.net/preview/2428804/page:8/
6. https://studopedia.ru/4\_15967\_kriterii-uspeshnosti-proekta.html
7. https://poisk-ru.ru/s60156t9.html
8. http://www.borlpasc.narod.ru/docym/yheb/yrok11.htm
9. https://qa-guide.ru/forums/topic/teorija\_testirovanija/
10. Интернет-курс «Основы программирования на С# 3.0: ядро языка» на портале НОУ ИНТУИТ

# **Приложения.**

## **Приложение 1. Мнения участников команды о командной работе**

**Иванов Алексей (капитан команды):**

Я по ходу семестра приложил свою руку почти ко всем частям всех проектов, проектировал их, распределял обязанности, консультировал сокомандников и участвовал в написании непосредственно кода для всех трех подпроектов. Не без трудностей, но со временем я привык к своим обязанностям, к своей команде, и все это сделало из меня куда более качественного командного программиста и лидера и углубило познания в том, что мы называем программной инженерией, — все это безусловно окажет большое позитивное влияние на дальнейшую работу и жизнь. Был рад провести этот семестр в своей роли лидера и получить такой прекрасный опыт. Горд за своих подчиненных, которые всегда старались, всегда слушали меня и с которыми мы достигли определенных успехов. Горд и счастлив быть капитаном своей команды.

**Аванесян Григорий (разработчик библиотеки классов):**

Мне очень понравилось разрабатывать проекты со своей командой. Я осознал значимость правильного распределения задач, научился писать многие методы и работать со многими функциям. От работы в своем коллективе получил только положительные впечатления и эмоции. Хотел бы и дальше работать в таком прекрасном коллективе, как «Калибр».

**Соболев Роман (разработчик библиотеки классов):**

Мне действительно понравилось работать в команде, это не только сплачивает коллектив, но и развивает тебя быстрее, чем работа над одиночным проектом. Все члены команды довольно ответственные и трудолюбивые. Также мне понравилось разрабатывать библиотеку классов для каждого из наших проектов, благодаря этому я узнал о множестве качественных алгоритмов. В будущем, работая в команде, я бы хотел попробовать себя в качестве тестировщика или разработчика интерфейсного проекта, чтобы более углубленно изучить эти структуры решения проекта.

**Борздов Марк (Тестировщик и отладчик, отчасти разработка библиотеки классов):**

Должен признать, что мне очень повезло с командой. Грамотное руководство -наш капитан - эффективно распределяло задачи, а остальные участники легко и быстро их выполняли. Недопонимания не возникали от слова совсем, из-за чего работать было одно удовольствие. Каждый из нас многому научился и усвоил пару-тройку уроков. Хочется отметить, что в работа в команде очень помогла нам сплотиться и получше узнать друг друга, так что мы научились не только программировать, но и понимать друг друга. Я бы очень хотел и дальше работать в нашем прекрасном коллективе.

**Мельников Михаил (Разработчик интерфейс-проекта):**

Мне понравилось работать в команде. Работая в ней, я имел возможность обмениваться знаниями и опытом с другими членами команды. Это позволяет нам учиться друг у друга, улучшать свои навыки и узнавать новые подходы к решению задач. Каждый член команды имеет свои навыки и сильные стороны. Распределение задач в команде позволяет нам использовать наши навыки максимально эффективно и получать результаты, которые были бы трудно достичь, работая самостоятельно. В большинстве случаев, когда возникают проблемы, их можно решить быстрее и эффективнее коллективно, исходя из мнения и опыта каждого участника команды. Взаимодействие и работа в команде позволяют нам распределять ответственность, координировать действия и увеличивать производительность работы.

**Запецкий Вадим (Новичок, обучающийся работе в командных проектах):**

Моя работа в команде программистов, где я новичок, была вдохновляющим опытом. Получение заданий от Алексея для изучения C# и поддержка от команды помогли мне быстро влиться в коллектив. Командная работа способствовала эффективному решению задач и достижению общих целей.

## **Приложение 2. «О распределении обязанностей при написании курсовой»**

**Глава 1. Вводная часть.**

Соболев Роман (пункты 1-3, 7)

Запецкий Вадим (пункты 4-6)

**Глава 2. Практическая часть.**

Мельников Михаил (пункт 1)

Аванесян Григорий (пункты 2,4)

Борздов Марк (пункты 3, 5)

Соболев Роман (пункт 5)

**Заключение**

Иванов Алексей

**Организационная деятельность, корректировка**

Иванов Алексей