МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

О.К. ГОЛОВНИН, А.А. СТОЛБОВА

ВВЕДЕНИЕ В СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И ОСНОВЫ ЖИЗНЕННОГО ЦИКЛА СИСТЕМНЫХ ПРОГРАММ

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве учебного пособия для обучающихся по основной образовательной программе высшего образования по направлению подготовки 09.03.01 Информатика и вычислительная техника

С А М А Р А
Издательство Самарского университета
2021

УДК 004.9(075)
ББК 22.18я7
Г612

Рецензенты: д-р техн. наук, проф. А. В. И в а щ е н к о,
канд. техн. наук, доц. О.Н. С а п р ы к и н

Головнин, Олег Константинович

Г612 **Введение в системное программирование и основы жизненного цикла системных программ**: учебное пособие / *О.К. Головнин, А.А. Столбова*. – Самара: Изд-во Самарского университета, 2021. – 172 с.: ил.

ISBN 978-5-7883-1695-6

Учебное пособие содержит информацию об основах системного программирования. Рассмотрены общие вопросы создания системных программ различных классов, описаны принципы оптимизации и особенности выполнения системных программ, отдельно рассмотрены вопросы построения трансляторов и компиляторов. Подробно рассмотрены современные технологии и особенности применения языков высокого уровня в процессе разработки системных программ. Представлены материалы для проведения практических занятий и задания для выполнения лабораторных работ. Даются рекомендации по стилю и технологии программирования, оформлению и документированию системных программ.

Предназначено для студентов высших учебных заведений, обучающихся по направлению 09.03.01 Информатика и вычислительная техника.

Подготовлено на кафедре информационных систем и технологий.

УДК 004.9(075)
ББК 22.18я7

ISBN 978-5-7883-1695-6
© Самарский университет, 2021

ОГЛАВЛЕНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 5

ВВЕДЕНИЕ 6

1 ОСНОВНОЙ ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ КУРСА 8

1.1 Введение в системное программирование 8
1.2 Системное программное обеспечение 11
1.3 Трансляторы 14
1.4 Компиляция 16
1.5 Оптимизация программ 19
1.6 Системы программирования 26
1.7 Архитектура системных программ 28
1.8 Особенности выполнения программ 31
1.9 Обработка ошибок 35

2 ПРАКТИЧЕСКИЕ АСПЕКТЫ СИСТЕМНОГО ПРОГРАММИРОВАНИЯ 38

2.1 Инструментальные средства разработки программ 38
2.2 Проектирование системного программного обеспечения 41
2.3 Принципы SOLID и паттерны проектирования 45
2.4 Лямбда-исчисление и язык интегрированных запросов 48
2.5 Эффективность алгоритмов и оптимизация кода 62
2.6 Тестирование, отладка, структурная обработка исключений 69
2.7 Прочие вопросы 71

3 ЛАБОРАТОРНЫЙ ПРАКТИКУМ 72

3.1 Общие требования к разрабатываемому программному обеспечению 72
3.2 Порядок выполнения лабораторных работ 73
3.3 Индивидуальные варианты 74

3.4 Анализ требований к системному программному обеспечению 78
3.5 Проектирование системного программного обеспечения 81
3.6 Создание сложной структуры данных 82
3.7 Разработка системного программного обеспечения с использованием принципов SOLID и паттернов проектирования 83
3.8 Вызов ассемблерных функций из языка высокого уровня 84
3.9 Организация доступа к данным путем объектно-реляционного отображения 88
3.10 Внедрение структурной обработки исключений 95
3.11 Оценка эффективности функционирования системного программного обеспечения 96
3.12 Документирование системного программного обеспечения 97

СПИСОК ЛИТЕРАТУРЫ 100

ПРИЛОЖЕНИЕ А Список теоретических вопросов к экзамену 103
ПРИЛОЖЕНИЕ Б Задание на контрольную работу 105
ПРИЛОЖЕНИЕ В Пример оформления отчета 108

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ОС – операционная система.

ПО – программное обеспечение.

СПО – системное программное обеспечение.

СУБД – система управления базами данных.

DLL – динамически подключаемая библиотека.

IDE – интегрированная среда разработки.

LINQ – язык интегрированных запросов.

SOLID – 5 основных принципов проектирования в объектно-ориентированном проектировании.

ВВЕДЕНИЕ

Изучение системного программирования является неотъемлемой частью подготовки специалистов в области информационных технологий и вычислительной техники.

Целью освоения дисциплины является подготовка квалифицированных специалистов, владеющих фундаментальными знаниями и практическими навыками в области выбора, проектирования, разработки, внедрения, оценки качества и анализа эффективности системного программного обеспечения (СПО).

Задачи дисциплины:

- ознакомление с основами функционирования и устройства СПО;
- обучение систематизированному представлению о принципах анализа, проектирования, разработки СПО;
- приобретение соответствующих практических навыков проектирования и программирования с использованием современных автоматизированных средств проектирования, разработки, совместной работы и поддержки проектов;
- приобретение навыков работы с языками программирования для создания СПО;
- получение практических навыков использования современных инструментальных средств, стандартных библиотек классов и шаблонов;
- формирование знаний и умений разработки СПО на объектно-ориентированных языках программирования;
- получение практических навыков по разработке и оформлению проектной и рабочей технической документации на СПО.

При изучении дисциплины с использованием материалов настоящего учебного пособия следует придерживаться следующих методических рекомендаций. Курс необходимо изучать последовательно и систематически, перерывы в занятиях и перегрузки нежелательны. Теоретический материал должен быть глубоко усвоен. Следует избегать механического запоминания отдельных положений. Студент должен разобраться в теоретическом материале и уметь применить его к решению практических задач и задач лабораторного практикума.

Учебное пособие состоит из трех разделов: в первом кратко рассмотрен теоретический материал курса, во втором – практические аспекты системного программирования, в третьем – особенности реализации СПО на современных языках программирования с использованием передовых инструментальных средств. Соответственно, первый раздел соответствует курсу лекций, второй – практическим занятиям, третий – лабораторному практикуму.

Каждый раздел содержит краткое содержание материала (выделен *курсивом*), который должен быть обозначен преподавателем и усвоен обещающимся по тематике раздела. Список литературы для изучения по курсу «Системное программирование» приведен в конце учебного пособия. В приложениях к учебному пособию приведены вопросы к экзамену и задания на контрольную работу.

Учебное пособие предназначено для студентов высших учебных заведений, обучающихся по направлению 09.03.01 Информатика и вычислительная техника, а также для всех желающих получить базовые навыки системного программирования при решении инженерных задач.

1 ОСНОВНОЙ ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ КУРСА

1.1 Введение в системное программирование

Рассматривается предмет изучения курса «Системное программирование», его связь с другими дисциплинами. Даются краткие сведения о содержании теоретического материала, лабораторных работ и практических занятий, обозначаются требования к текущей и промежуточной аттестации обучающихся по курсу. Рассматривается литература по курсу. Дается понятие системной программы, рассматриваются основные определения и термины. Рассматривается классификация и структура системного программного обеспечения.

Рассмотрим классификацию программного обеспечения. На сегодняшний день выделяют по меньшей мере три класса программного обеспечения (ПО): прикладное, промежуточное и системное.

Прикладное ПО – компьютерные программы, предназначенные для решения функциональных задач в определенной предметной области.

Это самый многочисленный класс программных продуктов.

Примеры прикладного ПО: текстовые редакторы (Microsoft Word, Libre Office), табличные редакторы (Microsoft Excel), графические редакторы (Adobe Photoshop).

Промежуточное (связующее) ПО – совокупность программ, осуществляющих управление программными ресурсами, порожденными программами и ориентированными на решение широкого класса задач.

Примеры промежуточного ПО:

- системы управления базами данных (СУБД);
- модули управления языком интерфейса информационных систем;

– программы сбора и предварительной обработки информации.

Промежуточное ПО часто оказывается скрытым в составе драйверов устройств или поставляется в виде библиотек для программирования. Поэтому такое ПО часто относят к системному программному обеспечению.

Существует и другой взгляд на промежуточное ПО, в котором оно представляет собой слой или комплекс технологического программного обеспечения для обеспечения взаимодействия между различными приложениями, системами, компонентами.

Примеры: веб-сервер, сервер приложений, сервисная шина, система управления контентом.

Системное ПО – совокупность программ, предназначенных для поддержания работоспособности системы обработки информации и управления (компьютера и компьютерных сетей) или повышения эффективности ее использования.

Функции системного ПО:

– создание операционной среды функционирования для программ;
– автоматизация разработки новых программ;
– обеспечение надежной и эффективной работы компьютера и компьютерной сети;
– проведение диагностики и профилактики аппаратуры компьютера и компьютерных сетей;
– выполнения вспомогательных технологических процессов (копирование, архивирование, восстановление после сбоев и т.п.).

Системное ПО тесно связано с типом компьютера и является его неотъемлемой частью. Системное ПО ориентировано на квалифицированных пользователей и профессионалов в области информационных технологий.

К системному ПО предъявляются высокие требования по надежности и технологичности работы, удобству и эффективности использования.

Существуют следующие группы системного ПО:

– операционные системы (ОС);
– интерфейсные оболочки ОС;
– системы управления файлами;
– системы программирования;
– утилиты;
– драйверы;
– средства сетевого доступа.

Системное программирование – это процесс разработки системных программ.

Другой взгляд на системное программирование – разработка программ сложной структуры.

Значительная часть системного и практически все прикладное ПО пишется на языках высокого уровня, что обеспечивает сокращение расходов на их разработку, модификацию и переносимость. Однако, разработка системного ПО предполагает знание и использование ассемблера для создания модулей и ассемблерных вставок.

Вопросы для самоконтроля

1) Что представляет собой прикладное ПО? Приведите примеры прокладного ПО.

2) Что представляет собой промежуточное ПО? Приведите примеры промежуточного ПО.

3) Что представляет собой системное ПО?

4) Какие функции выполняет системное ПО?

5) Какие группы системного ПО существуют?

6) Дайте определение системного программирования.

1.2 Системное программное обеспечение

Рассматриваются различные способы классификация системного программного обеспечения. Даются краткие сведения по следующим видам системного программного обеспечения: операционные системы, загрузчики, драйверы, системы программирования, трансляторы, компиляторы и интерпретаторы, отладчики, утилиты.

Существуют несколько способов классификации системного ПО. Ниже рассматриваются основные.

Традиционная классификация системного ПО:

- управляющее ПО – организует корректное функционирование всех процессов ОС и устройств компьютера;
- обрабатывающее ПО – обеспечивает выполнение специальных задач.

Управляющее системное ПО осуществляет оркестрирование вычислительных процессов, организует работу с внутренними данными ОС. Этот вид ПО располагается в основной памяти и представляет собой резидентные программы ОС. Системное управляющее ПО поставляется в виде инсталляционных пакетов ОС и драйверов устройств.

Обрабатывающее системное ПО функционирует в операционной среде ОС с целью выполнению специальных задач. Системное обрабатывающее ПО поставляется в виде дистрибутивных пакетов с программами инсталляции, разворачивающими ПО на конечном устройстве.

Альтернативная классификация системного ПО:

- базовое ПО – минимальный набор программных средств, обеспечивающих работу компьютера и компьютерной сети;

– сервисное ПО – программы и программные комплексы, которые расширяют возможности базового ПО и организуют удобную среду для работы других программ и пользователя.

Примеры базового ПО:

– операционные системы и драйверы в составе ОС;
– загрузчики;
– интерфейсные оболочки ОС;
– системы управления файлами.

Операционная система – совокупность программных средств, обеспечивающая управление аппаратной частью компьютера и прикладными программами, а также их взаимодействием между собой и пользователем. Примеры ОС: Windows, Linux, Unix, Android, iOS.

Загрузчик – программа, обеспечивающая выбор ОС при запуске компьютера.

Интерфейсные оболочки ОС предназначены для смены режима взаимодействия пользователя с прикладными программами и ресурсами компьютера. Они могут функционировать в текстовом или графическом режимах. Примеры интерфейсных оболочек: KDE, Gnome.

Системы управления файлами выделяются в отдельную группу системного ПО, однако любая система управления файлами разработана для работы в конкретной ОС и с конкретной файловой системой. Системы управления файлами предназначены для организации более удобного доступа к данным, когда вместо низкоуровневого доступа к данным с указанием конкретных физических адресов используется логический доступ с указанием имени файла.

Примеры сервисного ПО:

– драйверы специальных устройств, которые поставляются отдельно от ОС;
– программы диагностики работоспособности компьютера;
– антивирусные программы;

– программы обслуживания дисков;
– программы архивирования данных.

Сервисное ПО часто называются утилитами. **Утилиты** – программы, служащие для выполнения вспомогательных операций обработки данных или обслуживания компьютеров.

Примеры утилит: CCleaner, 7-Zip, WinRAR, AIDA, Disk Commander.

Отдельной крупной группой системного ПО, относящегося к обрабатывающему ПО или сервисному ПО, являются системы программирования (подробно рассматриваются в разделах 1.6 и 2.1).

Система программирования – набор специализированных программ, которые выступают инструментальными средствами разработчика для полной поддержки процессов совместной разработки, доступа к коду, проектирования, разработки, отладки и тестирования создаваемых программ, их развертывания.

Система программирования включает следующие средства:

– редактор текста – программа для ввода и модификации кода программы;
– транслятор – программа для преобразования программ, написанных на одном языке программирования, в программы на другом языке;
– компоновщик – редактор связей, объединяющий отдельные модули в единые программы, готовые к выполнению;
– отладчик – инструмент для поиска и устранения ошибок;
– библиотеки подпрограмм – дополнительные модули, упрощающие разработку.

Вопросы для самоконтроля

1) На какие классы разделяют системное ПО в рамках традиционной классификации?

2) На какие классы разделяют системное ПО в рамках альтернативной классификации?

3) Приведите примеры базового ПО.

4) Дайте определение операционной системы.

5) Для чего предназначены интерфейсные оболочки ОС?

6) Что представляют собой системы управления файлами? Для чего они предназначены?

7) Приведите примеры сервисного ПО.

8) Что представляет собой современная система программирования?

9) Какие средства включает в себя система программирования?

1.3 Трансляторы

Рассматриваются трансляторы и их виды: ассемблеры, компиляторы и интерпретаторы. Дается общая схема работы трансляторов. Рассматривается назначение и принципы построения трансляторов.

Транслятор – системная программа, преобразующая исходную программу на одном языке программирования в программу на другом языке.

Общая схема работы транслятора приведена на рисунке 1.

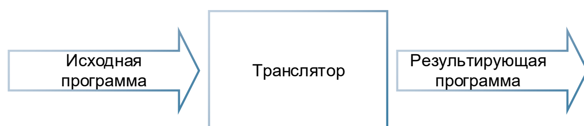

Рисунок 1 – Процесс трансляции

Трансляторы разделяются на три основных класса: ассемблеры, компиляторы и интерпретаторы.

Ассемблер – системная программа, которая преобразует символические конструкции в команды машинного языка. Под машинным языком тут и далее понимается язык конкретного компьютера (процессора) или группы компьютеров (семейства процессоров). Ассемблер осуществляет дословную трансляцию одной символической команды в одну машинную. Ассемблеры обеспечивают повышение эффективности программирования за счет того, что программист легче запоминает и воспринимает мнемоническое обозначение машинных команд, а не их двоичный код.

Компилятор – системная программа, выполняющая трансляцию программы на исходном языке программирования в программу на машинном языке. В связи с тем, что команды исходного языка значительно отличаются от команд машинного языка, одна команда исходного языка может транслироваться в 5-10 и более машинных команд. Процесс трансляции с таких языков обычно называется компиляцией, а исходные языки обычно относятся к языкам программирования высокого уровня.

Интерпретатор – системная программа, осуществляющая пооператорную трансляцию и выполнение исходной программы. В отличие от компилятора, интерпретатор не порождает на выходе программу на машинном языке, распознав команду исходного языка, он тут же выполняет ее. Интерпретатор позволяет начать обработку данных после написания даже одной команды, что делает процесс разработки и отладки программ более гибким. Отсутствие выходного машинного кода позволяет не порождать дополнительные файлы, а сам интерпретатор можно достаточно легко адаптировать к любым машинным архитектурам. У интерпретаторов два основных недостатка: первый – относительно низкая скорость работы интерпретируемых программ, второй – исходный код программы остается доступен конечным пользователям.

Существуют и другие виды системного программного обеспечения, непосредственно связанные с процессом трансляции.

Эмулятор – системная программа, обеспечивающая возможность без перекомпиляции выполнять на данном компьютере программу, которая для данного компьютера не предназначена. Эмуляторы используются достаточно часто в самых различных целях, например, для выполнения старых программ на новых компьютерах или для запуска приложений ОС Windows из ОС MacOS.

Перекодировщик – системная программа, переводящая программы, написанные на одном машинном языке, в программы на другом машинном языке.

Макропроцессор – системная программа, обеспечивающая замену одной последовательности символов другой. Макропроцессоры часто используются как надстройки над языками программирования, увеличивая функциональные возможности систем программирования.

Вопросы для самоконтроля

1) Дайте определение транслятора.
2) Что представляет собой ассемблер?
3) Дайте определение компилятора.
4) Дайте определение интерпретатора.
5) В чем отличие интерпретатора от компилятора?

1.4 Компиляция

Рассматривается процесс компиляции программы. Даются подробные сведения об основных фазах компиляции программы: лексический, синтаксический и семантический анализ, оптимизация, генерация кода, сборка.

Процесс компиляции разделяется на этапы, которые, в свою очередь, разделяются на фазы.

Этапы компиляции:

0. Предварительная обработка – присоединение исходных файлов, работа макросов (примеры: T4, шаблоны).
1. Анализ – определение структуры и значения исходного кода.
2. Синтез – построение целевого кода.

Выделяются три **фазы на этапе анализа**:

1. Лексический анализ – переход от последовательности знаков к символам языка. Лексический анализ похож на процесс чтения человеком текста программы. На этом этапе происходит обработка пробелов и удаление комментариев. Лексическому анализатору не важен смысл текста. Пример средства автоматизации: генератор лексических анализаторов Lex.
2. Синтаксический анализ (разбор) – определение общей структуры программы с пониманием порядка следования символов. Основа для работы синтаксического анализатора – синтаксические правила (грамматика). Синтаксический анализатор осуществляет обработку с привязкой к контексту. Результат работы синтаксического анализатора – синтаксическое дерево (пример – на рисунке 2). Существует множество методов синтаксического анализа, например метод рекурсивного спуска, метод восходящего анализа SLR и др. Пример средства автоматизации: генератор синтаксических анализаторов YACC.
3. Семантический анализ – определение смыслового значения текста программы и проверка особых свойств (например, типов переменных и областей их видимости).

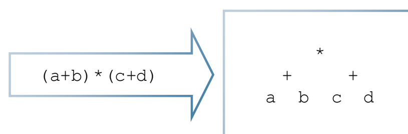

Рисунок 2 – Синтаксическое дерево

Выделяются пять **фаз на этапе синтеза**:

1. Генерация машинно-независимого кода – формирование кода на промежуточном языке, который не зависит от конечного компьютера;
2. Оптимизация машинно-независимого кода – повышение эффективности кода (подробно рассмотрено в разделе 1.5);
3. Распределение памяти – назначение каждой переменной адреса в одной из областей памяти:
   - статическая – время жизни переменной связано с временем жизни программы;
   - динамическая – время жизни переменной связано с временем жизни программного блока;
   - глобальная – время жизни переменной неизвестно.
4. Генерация машинного кода – формирование кода на машинном языке;
5. Оптимизация машинного кода – повышение эффективности целевого кода (кратко рассмотрено в разделе 1.5).

Вопросы для самоконтроля

1) На какие этапы разделяют процесс компиляции?
2) Какие фазы анализа выделяют?
3) Что представляет собой лексический анализ?
4) Что представляет собой синтаксический анализ?

5) Что представляет собой семантический анализ?
6) Какие фазы синтеза выделяют?

1.5 Оптимизация программ

Подробно рассматриваются вопросы машинно-независимой оптимизация программ: удаление недостижимого кода, оптимизация линейных участков программы, свертка, перестановка операций, арифметические преобразования, оптимизация вычисления логических выражений, оптимизация циклов. Даются краткие сведения о машинно-зависимой оптимизации программ.

Оптимизация – процесс создания эффективного целевого кода.

Критерии эффективности:

– время выполнения программы;
– объем используемой памяти в процессе выполнения программы;
– объем (размер) программы;
– равномерность загрузки оборудования.

Оптимизацию не требуется проводить в случаях, если:

– программа работает «хорошо»: быстро, не использует лишнюю оперативную память, занимает мало места на жестком диске;
– программа обладает малым временем жизни (например, студенческие работы, программы для научных исследований).

Во всех остальных случаях стоит рассмотреть возможность оптимизации программы. Оптимизация требует затрат времени программиста или компилятора, проводящего ее. Быструю оптимизацию лучше делать всегда, медленную – в редких случаях. Среды компиляции имеют надстройки, позволяющие включать и отклю-

чать оптимизацию при компиляции, поскольку при включении оптимизации затрудняется отладка.

Виды оптимизации:

– ручная – выполняется программистом на исходном языке программирования;
– машинно-независимая оптимизация – преобразование компиляторам программы на промежуточном языке;
– машинно-зависимая оптимизация (оптимизация машинного кода) – преобразование программы компилятором на выходном языке.

Машинно-независимая оптимизация заключается, как правило, в выполнении следующих действий.

Подстановка значений констант:

```
// Исходный код:
const int c1 = 5;
static readonly int c2 = 5;
...
int res = c1 + c2;
// Оптимизированный код:
static readonly int c2 = 5;
...
int res = 5 + c2;
```

Вычисление выражений (не выполняется при указании volatile):

```
// Исходный код:
int i = 5;
int b = 6;
int c = i*b;
```

```
// Оптимизированный код:
int c = 30;
```

Удаление недостижимого кода:

```
// Исходный код:
if (true) Console.Write("true") else Console.Write("false");
// Оптимизированный код:
Console.Write("true");
```

Арифметические преобразования:

```
// Исходный код:
a = b*c + b*d;
e = e * 0;
// Оптимизированный код:
a = b*(c+d);
e = 0;
```

Устранение избыточных вычислений:

```
// Исходный код:
d = d + b*c;
a = d + b*c;
c = d + b*c;
// Оптимизированный код:
t = b*c;
d = d + t;
a = d + t;
c = a;
```

Удаление ненужных присваиваний:

```
// Исходный код:
a = b*c;
d = b + c;
a = d*c;
// Оптимизированный код:
d = b +c;
a = d*c;
```

Оптимизация вычисления логических выражений:

```
// Исходный код:
a | b | c | d
a & b & c & d
// Оптимизированный код:
a || b || c || d
a && b && c && d
```

Подстановка пода функции вместо ее вызова:

```
// Исходный код:
int f(int a, int b, int c, int d)
{
    return a*b*c*d;
}
...
int res = a + b + c + d + f(a, b, c, d);
// Оптимизированный код:
int res = a + b + c + d + a*b*c*d;
```

Вынесение инвариантных вычислений из тела цикла:

```
// Исходный код:
for (int i = 0; i < 1000; i++)
    for (int j = 0; j < 1000; j++)
            a[i, j] = a*b*c + i + j;
// Оптимизированный код:
int t1 = a*b*c;
for (int i = 0; i < 1000; i++)
{
    int t2 = t1 + i;
    for (int j = 0; j < 1000; j++)
            a[i, j] = t2 + j;
}
```

Замена операций с переменными цикла (индукция):

```
// Исходный код:
int s = 10;
for (int i = 0; i < 1000; i++)
    a[i] = s*(i+1);
// Оптимизированный код:
int s = 10;
int t = 0;
for (int i = 0; i < 1000; i++)
{
    t = t + s;
    a[i] = t;
}
```

Замена операций с переменными цикла (элиминация):

```
// Исходный код:
int s = 10;
for (int i = 0; i < 1000; i++)
{
    q = q + f(s);
    s = s + 10;
}
// Оптимизированный код:
int s = 10;
int m = s + 10*1000;
while (s <= m)
{
    q = q + f(s);
    s = s + 10;
}
```

Слияние циклов:

```
// Исходный код:
for (int i = 0; i < 1000; i++)
{
    f1(i);
}
for (int j = 0; j < 1000; j++)
{
    f2(j);
}
// Оптимизированный код:
for (int i = 0; i < 1000; i++)
```

```
{
    f1(i);
    f2(i);
}
```

Расщепление циклов:

```
// Исходный код:
for (int i = 0; i < 1000; i++)
{
    if (x > y) f1(i)
            else f2(i);
}
// Оптимизированный код:
if (x > y)
    for (int i = 0; i < 1000; i++)
            f1(i);
else
    for (int i = 0; i < 1000; i++)
            f2(i);
```

Кроме перечисленного, есть и другие виды машинно-независимой оптимизации, например, передача в функцию параметров не через стек, а через регистр или ссылки.

Машинно-зависимая оптимизация заключается, как правило, в выполнении следующих действий:

- замена медленных команд на быстрые;
- замена общих команд на команды, специфичные для текущего процессора;
- распределение регистров;

– учет технических особенностей компьютера при распределении нагрузки.

Вопросы для самоконтроля

1) Что представляет собой процесс оптимизации?
2) Какие критерии эффективности существуют?
3) В каких случаях не требуется проводить оптимизацию?
4) Какие виды оптимизации выделяют?
5) Приведите примеры действий, проводимых в рамках машинно-независимой оптимизации.
6) Какие действия выполняются в рамках машинно-зависимой оптимизации?

1.6 Системы программирования

Рассматриваются структура и функциональные возможности современных систем программирования.

Система программирования может быть представлена набором независимых инструментов или полноценной интегрированной средой разработки. Современные системы программирования объединяют в себе (интегрируют) следующие компоненты:

– редакторы текстов с подсветкой синтаксиса;
– средства тестирования и отладки;
– средства компиляции и поставки (развертывания);
– средства документирования;
– средства профилирования;
– средства организации командной работы;
– средства доступа к репозиториям кода;
– конструкторы визуальных интерфейсов;
– интерактивные инструменты получения справочной информации;

– средства работы с источниками данных и серверами;
– инструменты реинжиниринга;
– диспетчеры пакетов;
– эмуляторы;
– редакторы ресурсов;
– средства рефакторинга;
– средства отладки и просмотра в режиме ассемблера.

Функциональные возможности редактора текста с подсветкой синтаксиса:

– редактирование текста программы: создание, редактирование, сохранение файлов с текстом программы;
– поддержка многооконной работы с возможностью управления вкладками;
– настройка сочетания клавиш и шаблонов;
– интеграция с компилятором и средствами статического анализа кода:
    - визуализация текста с выделением лексем;
    - дополнение кода, интерактивная подсказка;
    - всплывающие подсказки;
    - отображение ошибок;
    - навигация по коду;
    - рефакторинг кода;
– интеграция с отладчиком:
    - отображение контрольных точек останова при отладке;
    - отображение текущего значения переменной.

Функциональные возможности средств тестирования и отладки:

– пошаговое выполнение программы;
– выполнение программы до точки останова;
– приостановка выполнения программы;

– просмотр и изменение значений переменных;
– вычисление выражений в текущем контексте;
– работа с точками останова;
– выдача информации в терминах исходной программы.

Особенности работы конкретных систем программирования и инструментов, встраиваемых в них, приведены в разделе 2.1.

Вопросы для самоконтроля

1) Какие компоненты объединяют в себе современные системы программирования?

2) Какие функциональные возможности предоставляет редактор текста с подсветкой синтаксиса?

3) Какие функциональные возможности предоставляют средства тестирования и отладки?

1.7 Архитектура системных программ

Рассматриваются основные архитектурные модели системного программного обеспечения, в том числе одно-, двух- и трехуровневые. Даются сведения об организации различных вариантов межпрограммных связей. Рассматриваются основные технологии, используемые для реализации межпрограммных связей. Даются сведения о библиотеках подпрограмм, используемых при разработке программного обеспечения. Рассматриваются вопросы создания и использования динамически подключаемых библиотек (DLL). Даются краткие сведения о диспетчерах пакетов, предназначенных для управления библиотеками подпрограмм и установления зависимостей между ними.

Архитектурная модель ПО – принципиальная организация ПО, воплощенная в его элементах, их взаимоотношениях друг с

другом и со средой, а также принципы, направляющие проектирование и эволюцию ПО.

Существует множество способов классификации и описания архитектурных моделей ПО, рассмотрим основные из них, где классификация производится по количеству звеньев (уровней):

- одноуровневая – все компоненты программы размещены на одном компьютере и на нем же выполняются;
- двухуровневая архитектура предлагает наличие двух компонентов: клиента и сервера (к которому подключен клиент).
- трехуровневая архитектура предполагает наличие трех компонентов: клиента, сервера приложений (к которому подключен клиент) и сервера баз данных (с которым работает сервер приложений).

Организация межпрограммных связей возможна многими способами. Связи могут быть установлены:

- на уровне библиотек (для программ, функционирующих на одном устройстве);
- на уровне сервисов, осуществляющих обмен данными по различным протоколам (для клиент-серверных приложений, для интеграции серверов);
- на уровне данных (совместный доступ к данным);
- через различные технологии интеграции (ESB, CORBA, COM, DCOM, ActiveX и др.).

Разработка СПО может быть значительно облегчена за счет использования **библиотек подпрограмм**, инкапсулирующих функции и визуальные компоненты для повторного использования.

Можно выделить следующие типы библиотек:

- библиотеки функций:
    - библиотеки для расширения возможностей языков программирования;

- библиотеки для решения задач в конкретной предметной области;

- библиотеки классов – аналогичны библиотекам функций, но ориентированы на объектно-ориентированное программирование;
- библиотеки компонентов – библиотеки готовых откомпилированных программных модулей, предназначенных для использования в качестве составных частей программ;

Динамически подключаемые библиотеки (DLL) подключаются к программе не во время компиляции программы, а непосредственно в ходе её выполнения. На этапе компоновки программы редактор связей формирует таблицу точек вызова функций DLL для последующей операции динамического связывания. Таким образом, процесс полной компоновки завершается уже на этапе выполнения целевой программы.

Преимущества такого подхода:

- не требуется включать в программу код часто используемых функций, что существенно сокращает объем кода;
- различные программы, выполняемые в некоторой ОС, могут пользоваться кодом одной и той же библиотеки;
- изменения и улучшения функций библиотек не требуют перекомпиляции программы.

В современных средах программирования для поставки и интеграции библиотек используются **диспетчеры пакетов**, предназначенные для управления библиотеками подпрограмм и установления зависимостей между ними. Таким образом, осуществляется внедрение необходимой версии библиотеки и ее непрерывная актуализация, при обновлении библиотеки поставщиком. При использовании диспетчера пакетов нет необходимости включать стронные библиотеки подпрограмм в исходный код и хранилище кода – они будут загружены при сборке программы.

Примеры диспетчеров пакетов: NuGet, Maven.

Вопросы для самоконтроля

1) Дайте определение архитектурной модели ПО.

2) Какие классы архитектурных моделей ПО выделяют при классификации по количеству звеньев?

3) Какими способами возможна организация межпрограммных связей?

4) Для чего применяются библиотеки подпрограмм?

5) Какие типы библиотек подпрограмм выделяют? В чем их отличие?

6) В чем заключается принцип работы динамически подключаемых библиотек?

7) Какие преимущества использования динамически подключаемых библиотек?

8) Для чего применяются диспетчеры пакетов? Приведите пример диспетчеров пакетов.

1.8 Особенности выполнения программ

Рассматриваются особенности выполнения программ в многопоточном режиме. Рассматривается работа с основными видами памяти: стеком и кучей. Рассматриваются особенности стека и кучи. Рассматриваются вопросы работы с ресурсами операционной системы, в частности, с файлами. Дается описание процесса профайлинга. Рассматривается профайлинг программ по использованию оперативной памяти и процессорного времени.

Однопоточный последовательный процесс имеет только один поток управления, который выполняется в том же адресном пространстве, что и процесс, породивший его.

Многопоточный процесс имеет несколько параллельно выполняющихся потоков, у каждого из которых выделен свой стек и обеспечивается хранение собственных значений регистров. Потоки

работают в общей основной памяти и используют адресное пространство процесса, породившего их.

Выполнение программы в многопоточном режиме дает следующие преимущества:

- увеличение скорости за счет на использования «облегченных» потоков, работающих в общем пространстве виртуальной памяти;
- сокращение объема используемой памяти за счет использования потоками общих ресурсов;
- сокращение времени на смену контекста потока за счет использования «облегченных» потоков.

Контекст потока – структура данных, описывающая состояние потока на момент последнего исполнения потока:

- программный счетчик, регистр состояния и содержимое регистров процессора;
- указатели на стек ядра и пользовательский стек;
- указатели на адресное пространство потока.

Контекст потока необходимо сохранять и восстанавливать при переключении потоков или при возникновении других событий, влияющих на его выполнение. Контекст потока, как правило, сохраняется в текущем стеке ядра потока.

Примеры в .NET: Thread, Task, Task.Factory.StartNew, Parallel LINQ.

В процессе работы СПО использует, в основном, два вида памяти: стек и кучу.

Стек – структура данных, организовывается по принципу LIFO (Last In – First Out). Доступ в стеке всегда есть только к последнему добавленному в стек элементу. В оперативной памяти каждому потоку выполнения выделяется определенный размер стека, т.е. размер стека – фиксированная величина. Превышение лимита приводит к переполнению стека – StackOverflowException.

Куча – хранилище данных, расположенное в оперативной памяти, которое допускает динамическое выделение памяти. Местоположение данных в куче определяется указателем-ссылкой IntPtr. Куча обеспечивает произвольный доступ к любому элементу, но каждый раз требуется вычислять физическое расположение данных в куче по указателю и типу данных.

Таким образом, стек – быстрый (всегда известен адрес последнего элемента), небольшой, в нем организован доступ только к последнему элементу. Куча – медленная (требуется вычислять адрес элемента), большая (ограничена размером оперативной памяти), в ней организован доступ к любому элементу по указателю.

Особенности в .NET: в общем случае структуры располагаются в стеке (если не используется упаковка), объекты – в куче.

ОС обеспечивает управление ресурсами (память, файлы, процессорное время, оборудование). Будем называть ресурсы компьютера, выделяемые ОС, **ресурсами ОС или объектами ядра ОС**.

Действия ОС по управлению ресурсами ОС:

- создание и удаление;
- планирование ресурсов;
- обеспечение взаимодействия;
- разрешение ошибочных и тупиковых ситуаций.

Объекты ядра представляет собой блок памяти, содержащий имя объекта, описатель или дескриптор объекта (Handle), класс защиты, счётчик количества пользователей и другую информацию (например, смещение при открытии файла и т.п.).

Примеры объектов ядра ОС Windows:

- процесс, поток;
- файл;
- файл в памяти;
- событие, семафор, мьютекс;
- канал, сокет и т.д.

ОС ведет учет объектов и управляет ими. Пользователь может запросить информацию об объекте. Объекты можно создавать, уничтожать, открывать и закрывать посредством системных вызовов ОС по просьбе внешней программы. Созданные объекты закрепляются (принадлежат) создавшим их процессам. При порождении дочернего процесса, объекты ядра (как правило) наследуются из родительского, при этом создается копия объекта, принадлежащая уже дочернему процессу. Например, так наследуются стандартные потоки ввода/вывода, открытые файлы, каналы, семафоры и др. Номер дескриптора (Handle) при наследовании сохраняется.

Особенности использования в .NET: для освобождения ресурсов ядра ОС необходимо использовать конструкцию using или блок finally для выполнения метода Dispose(), поскольку сборщик мусора не гарантирует освобождение неуправляемых ресурсов.

Профайлинг (профилирование) программы – сбор характеристик работы программы, проводимый с целью анализа работы и оптимизации программы.

Характеристики работы программы могут быть аппаратными (время работы, используемая память) или программными (особенности выполнения и частота вызовов функций). Профайлинг программ используется, чтобы определить те участки программы, которые работают неэффективно и могут быть оптимизированы.

Для выполнения профайлинга используются программы-профайлеры, например, профайлер процессорного времени JetBrains dotTrace, профайлер использования оперативной памяти JetBrains dotMemory.

С другой информацией по теме профилирования и оптимизации программ можно ознакомиться в разделе 2.5.

Вопросы для самоконтроля

1) Что представляет собой однопоточный режим выполнения программ?

2) Что представляет собой многопоточный режим выполнения программ?

3) Какими преимуществами обладает многопоточный режим выполнения программ?

4) Дайте определение контекста потока.

5) Какие данные содержатся в контексте потока?

6) Объясните принцип работы стека.

7) Объясните принцип работы кучи.

8) В чем отличия стека и кучи?

9) Какие действия выполняет ОС по управлению ресурсами?

10) Что представляют собой объекты ядра? Приведите примеры объектов ядра ОС Windows.

11) Дайте определение профайлинга программ. Для чего он используется?

12) Какие программы используются для профайлинга? Приведите примеры таких программ.

1.9 Обработка ошибок

Рассматриваются вопросы обработки ошибок при выполнении программ. Рассматриваются вопросы перехвата исключительных ситуаций и вопросы их корректной обработки в соответствии со структурой программы и политиками обработки исключений.

При появлении ошибок в программе во время выполнения происходит выброс исключения (Exception).

Примеры известных исключений:

- DivideByZeroException – возникает при делении на 0;
- IndexOutOfRangeException – индекс находится вне границ массива.

В зависимости от типа исключения, они могут или не могут быть обработаны программой. Пример исключения, которое не может быть обработано – StackOverflowException – переполнение стека.

Обработка исключений в современных языках программирования, как правило, осуществляется в соответствии со следующим шаблоном:

```
try
{
<Защищаемый код>
}
catch (<Имя исключения>)
{
<Обработчик исключения>
}
finally
{
<Завершение>
}
```

Секция кода «try» должна быть объявлена обязательно. Секция кода «catch» может повторятся несколько раз, может отсутствовать. Секция кода «finally» может быть объявлена только один раз, либо отсутствовать. В обработчике исключений обязательно должна присутствовать либо секция «catch», либо секция «finally», либо обе.

При возникновении ошибки в защищаемом коде происходит последовательная проверка блоков «catch» сверху вниз: выполняется поиск первого блока, который может обработать возникшее исключение. Именно поэтому блоки «catch» должны располагаться от наиболее специфичных к наиболее общим – при указании общего

блока первым он обработает все возникшие исключения. Секция кода «finally» выполнится в любом случае – не важно, возникла ошибка или нет.

Обработку ошибок следует проводить в соответствии с **политикой обработки исключений**, определяющей поведение программы при обработке ошибок. Политика обработки исключений зависит от структуры программы, поэтому такая обработка исключений называется структурной.

Пример – исключение возникло в программе при обращении к базе данных. В слое бизнес-логики программы, работающем с базой данных, нет возможности вывести сообщение пользователю, поскольку слой не имеет графического интерфейса. Сообщение об ошибке должно быть направлено на верхние уровни, при этом, например, сведения о некоторых ошибках базы данных требуется предоставлять разработчику, через системы отслеживания (Redmine, YouTrack, Bugzilla) или инструменты логирования (log4net, NLog). Поведение программы должно меняться без перекодирования, поэтому политики обработки исключений гибко настраиваются с помощью конфигурационных файлов.

Вопросы для самоконтроля

1) По какой причине происходит выброс исключений?
2) Приведите примеры известных исключений.
3) По какому шаблону, как правило, происходит обработка исключений?

2 ПРАКТИЧЕСКИЕ АСПЕКТЫ СИСТЕМНОГО ПРОГРАММИРОВАНИЯ

2.1 Инструментальные средства разработки программ

Рассматриваются назначение и функции инструментальных средств, используемых при разработке системных программ.

Интегрированная среда разработки (Integrated Development Environment, IDE) – набор инструментальных средств для разработки и отладки программ, имеющий общую интерактивную графическую оболочку.

Некоторые компоненты современных IDE:

- единая интерактивная оболочка, обеспечивающая вызов всех других компонент из основной среды;
- текстовый редактор для набора и редактирования исходных текстов программ;
- система сборки для компиляции и компоновки;
- отладчик;
- профилировщик – инструмент для накопления и анализа статистических данных, полученных в результате исполнения программы;
- средства проведения рефакторинга – инструментарий систематических групповых модификаций программ в среде без принципиальных изменений их функциональности с целью улучшения качества кода;
- генератор тестов – инструмент для генерации типовых тестов для тестирования модулей;
- система управления версиями исходных кодов или инструмент интеграции с одной из существующих версионных систем;

– инструменты поддержки командной разработки программ;
– инструменты анализа кода;
– инструменты визуализации сгенерированного бинарного кода;
– инструменты «запутывания» кода (обфускация);
– шаблонизаторы кода;
– инструменты генерации диаграмм в различных нотациях.

Сравнение некоторых современных IDE по функциональности и поддерживаемым языкам приведены в таблицах 1–3.

Таблица 1 – Функциональные возможности редактора текста

Характеристика	Visual Studio	IntelliJ Idea	Eclipse	NetBeans	CodeLite
Основные возможности					
Создание, просмотр, редактирование проектов и отдельных файлов программ	Да	Да	Да	Да	Да
Поддержка многооконной работы	Да	Да	Да	Да	Да
Настройка сочетания клавиш и шаблонов	Да	Да	Да	Да	Да
Интеграция с компилятором и средствами статического анализа кода					
Визуализация текста с выделением лексем	Да	Да	Да	Да	Да
Дополнение кода, интерактивная подсказка	Да	Да	Да	Да	Да
Всплывающие подсказки	Да	Да	Да	Да	Нет
Отображение ошибок	Да	Да	Да	Да	Да
Рефакторинг кода	Да	Да	Да	Да	Да
Интеграция с отладчиком					
Отображение текущего значения переменной	Да	Да	Не со всеми языками	Не со всеми языками	Да
Отображение контрольных точек останова при отладке	Да	Да	Не со всеми языками	Не со всеми языками	Да
Учет особенностей региона и выполняемой ОС					
Кроссплатформенность	Да	Да	Да	Да	Да
Мультиязычность	Да	Да	Да	Да	Да

Таблица 2 – Функциональные возможности средств отладки

Характеристика	Visual Studio	IntelliJ Idea	Eclipse	NetBeans	CodeLite
Пошаговое выполнение программы	Да	Да	Не со всеми языками	Не со всеми языками	Да
Выполнение программы до точки останова	Да	Да	Не со всеми языками	Не со всеми языками	Да
Приостановка выполнения программы	Да	Да	Не со всеми языками	Не со всеми языками	Да
Просмотр и изменение значений переменных	Да	Да	Не со всеми языками	Не со всеми языками	Да
Вычисление выражений в текущем контексте	Да	Да	Не со всеми языками	Не со всеми языками	Да
Выдача информации в терминах исходной программы	Да	Да	Не со всеми языками	Не со всеми языками	Да

Таблица 3 – Поддерживаемые языки программирования

Язык программирования	Visual Studio	IntelliJ Idea	Eclipse	NetBeans	CodeLite
C/C++	Да	Нет	Да	Да	Да
JAVA	Нет	Да	Да	Да	Нет
C#	Да	Нет	Нет	Нет	Нет
Python	Частично	Нет	Да	Да	Нет
Ruby	Нет	Да	Нет	Да	Нет
Groovy	Нет	Да	Да	Да	Нет
PHP	Нет	Нет	Да	Да	Нет
Assembler	Как вставки	Нет	Нет	Нет	Нет
JavaSctipt	В доп. пакете	Да	В доп. пакете	В доп. пакете	Нет
Pascal	Нет	Нет	Нет	Нет	Нет

IntelliJ IDEA

IntelliJ IDEA – интегрированная среда разработки программного обеспечения на Java от компании JetBrains. Поддерживаемые технологии:

- полнофункциональная среда разработки под JVM, при этом есть поддержка еще и различных языков: Java, PHP, JavaScript, HTML, CSS, SQL, Ruby, Python;
- поддержка технологий Java EE, Spring/Hibernate и других;
- внедрение и отладка с большинством серверов приложений.

Visual Studio

Интегрированная среда разработки Visual Studio разработана компанией Microsoft и поддерживает следующие языки разработки: C#, C++, F#, Python, HTML, CSS, JavaScript.

Имеющиеся компоненты:

- редакторы текстов с подсветкой синтаксиса;
- список задач;
- интерактивные инструменты получения справочной информации;
- средства тестирования и отладки;
- средства компиляции и поставки (развертывания);
- средства документирования;
- средства профилирования;
- средства организации командной работы;
- средства доступа к репозиториям кода;
- конструкторы визуальных интерфейсов;
- средства работы с источниками данных и серверами;
- инструменты реинжиниринга;
- диспетчеры пакетов;
- эмуляторы;
- редакторы ресурсов;
- средства рефакторинга;

2.2 Проектирование системного программного обеспечения

Рассматриваются вопросы проектирования системного программного обеспечения. Даются сведения об основных этапах проектирования. Даются краткие сведения о методологиях проектирования.

При разработке сложного системного ПО соблюдаются следующие **принципы** [7].

Принцип оптимальности предполагает оптимизацию решений, обеспечивающих максимальный, интенсивно увеличивающийся во времени прирост промышленного потенциала на основе достижений науки и техники.

Принцип агрегирования позволяет осуществить членение сложного многоуровневого ПО на компоненты с целью снижения размерности решаемых задач, а также поэтапного ввода объектов.

Принцип управляемости – возможность воздействовать на состояние ПО.

Принцип автоматизации обеспечивает оперативную и качественную переработку информации с воздействием в необходимых случаях на объект управления.

Принцип стандартизации гарантирует информационное единство, совместимость, инвариантность, преемственность проектных решений и др.

Принцип системного единства состоит в том, что на всех стадиях создания, функционирования и развития ПО целостность должна обеспечиваться связями между образующими компонентами, а также функционированием специальной подсистемы (системы) управления.

Принцип развития требует, чтобы любой компонент разрабатывался и функционировал как развивающаяся система, в которой предусмотрена возможность совершенствования компонентов и связей между ними на основе принципов агрегирования и стандартизации.

Принцип надежности предполагает работоспособность объекта, построенного в общем случае из ненадежных элементов, за счет специальных средств стабилизации заданных характеристик надежности ПО и его элементов.

Основные свойства сложного СПО:

- многомерность и иерархичность, обусловленная большим числом взаимосвязанных между собой элементов;
- эмерджентность;
- многофункциональность элементов;
- многокритериальность, обусловливаемая имманентностью (несовпадением) целей отдельных элементов;
- сложное (вероятностное и динамическое) поведение, проявляющееся во взаимосвязи подсистем и требующее обратной связи при управлении;
- сложностью информационных процессов;
- необходимостью высокой автоматизации.

Этапы построения сложного СПО:

- техническое задание;
- научно-исследовательская работа;
- эскизный проект;
- технический проект;
- рабочий проект;
- технология изготовления и испытания спроектированного объекта, внесения коррекции.

На этапе разработки технического задания решаются следующие задачи:

- поиск необходимой научно-технической информации;
- анализ выбранной информации, формулировка требований;
- перечисление функций;
- определение условий работоспособности и ограничений;
- определение требований к выходным параметрам;
- определение характеристик модулей;
- разработка алгоритмов.

На этапе научно-исследовательской работы необходимо решение следующих задач:

- формулирование критериев качества;
- управление экспериментами;
- проведение экспериментов, обработка результатов;
- разработка математических моделей и их идентификация;
- формирование обобщенного критерия качества;
- решение задачи оптимизации (обобщенный критерий – целевая функция);
- поиск принципиальной возможности построения системы;
- разработка новых технических средств, в том числе средств контроля и измерений.

На этапе эскизного проектирования производится решение следующих задач:

- разрабатывается эскиз проектируемой системы с детальной разработкой ее возможностей;
- принимаются предварительные проектные решения, оформляются проектные документы;
- производятся расчеты для разработки проектных документов.

На этапе технического проектирования детализируются и уточняются решения, принятые при эскизном проектировании. Большинство решений и документов, принятых и сформированных на этапах эскизного и технического проектирования, используются только для выполнения рабочего проектирования.

На этапе рабочего проектирования основным видом выполняемых работ является оформление проектных решений в виде чертежей (диаграмм) и спецификаций к ним. Современные средства вычислительной техники позволяют полностью автоматизировать оформление чертежей и спецификаций.

При проектировании технологии производят:

- поиск и выбор исходной информации (об объекте, подлежащем изготовлению);
- анализ и обработку данных в целях определения маршрутов обработки, последовательности технологических операций и режимов их проведения, потребности в инструменте и измерительном оборудовании, в создании специальной оснастки;
- оформление соответствующей технологической документации.

Проектирование СПО состоит из двух основных этапов:

- обоснование исходных данных (технического задания) для проектирования;
- проектирование СПО для сформулированных исходных данных.

2.3 Принципы SOLID и паттерны проектирования

Рассматриваются принципы SOLID. Даются примеры их реализации. Дается определение паттерна проектирования и классификация паттернов проектирования. Подробно рассматривается паттерн проектирования Model-View-Presenter на примере.

Существует 5 основных принципов проектирования в объектно-ориентированном проектировании (аббревиатура – **SOLID**).

Принцип единственной обязанности (Single responsibility principle) – каждый объект должен иметь одну обязанность и эта обязанность должна быть полностью инкапсулирована в класс, все его сервисы должны быть направлены на обеспечение этой обязанности.

Принцип открытости-закрытости (Open/closed principle) – программные сущности (классы, модули, функции и др.) должны быть открыты для расширения, но закрыты для изменения.

Принцип подстановки Барбары Лисков (Liskov substitution principle) – функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом.

Принцип разделения интерфейсов (Interface segregation principle) – много специализированных интерфейсов лучше, чем один универсальный.

Принцип инверсии зависимостей (Dependency inversion principle) состоит из двух положений:

- модули верхних уровней не должны зависеть от модулей нижних уровней, оба типа модулей должны зависеть от абстракций;
- абстракции не должны зависеть от деталей, детали должны зависеть от абстракций.

Паттерн (шаблон) проектирования – это формализованное описание часто встречающейся задачи проектирования, удачное решение данной задачи, а также рекомендации по применению этого решения в различных ситуациях. Паттерн именует, абстрагирует и идентифицирует ключевые аспекты структуры общего решения, которые и позволяют применять его для создания повторно-используемого решения.

По назначению паттерны можно разделить на три класса:

- порождающие паттерны;
- структурные паттерны;
- паттерны поведения.

Паттерн **P** описывается тетрадой **<N, G, S, R>**, где:

N – уникальное имя паттерна, позволяющее однозначно идентифицировать паттерн, ссылаться на него, увеличить уровень абстрагирования, путем включения в имя паттерна описание проблемы проектирования, решение проблемы и последствия применения паттерна;

G – задача и условия применения паттерна, описываемые в терминах естественного языка и предикатов;

S – решение задачи с помощью обобщенного сочетания элементов, отношений, способов взаимодействия;

R – результаты применения паттерна, его влияние на объект применения паттерна.

Рассмотрим один из паттернов – *паттерн «Модель-Вид-Представитель» (MVP)*.

Задача: отделить друг от друга логику работы программы, отображение данных, сами данные.

Решение: вводится три сущности (рисунок 3):

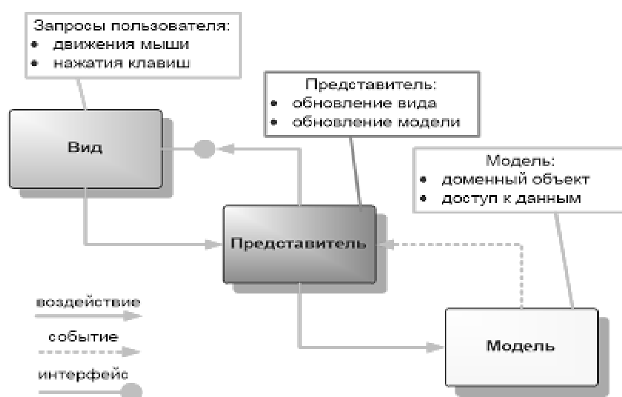

Рисунок 3 – Схема паттерна «Модель-Вид-Представитель»

- «Модель» – является источником (хранилищем) данных;
- «Вид» – сущность, отображающая модель. В случае необходимости каких-либо действий (логики), «Вид» обращается к «Представителю»;
- «Представитель» – содержит в себе всю логику работы, отвечает за синхронизацию «Вида» и «Модели». Когда

«Вид» сообщает ему о событии, он обновляет «Модель» и производит синхронизацию (если необходимо).

2.4 Лямбда-исчисление и язык интегрированных запросов

Рассматриваются вопросы применения лямбда-исчисления и логики предикатов в программировании на языке высокого уровня. Даются основные методы языка интегрированных запросов LINQ. Рассматриваются примеры использования LINQ.

Язык интегрированных запросов LINQ – это набор появившихся в Visual Studio 2008 функций, которые значительно расширяют возможности синтаксиса языков C# и Visual Basic. LINQ предлагает стандартные, легко запоминающиеся шаблоны для выполнения запросов и обновления данных, и эта технология может быть расширена с целью поддержки теоретически любого типа хранилища данных. Visual Studio содержит сборки поставщиков LINQ, позволяющие использовать LINQ с коллекциями .NET Framework, базами данных SQL Server, объектами DataSet ADO.NET и XML-документами.

Традиционно запросы к данным выражаются в виде простых строк без проверки типов при компиляции или поддержке IntelliSense. Кроме того, разработчику приходится изучать различные языки запросов для каждого из типов источников данных: баз данных SQL, XML-документов, различных веб-служб и т. д. LINQ делает запросы очень удобной конструкцией языков C# и Visual Basic. Разработчики создают запросы к строго типизированным коллекциям объектов с помощью зарезервированных слов языка и знакомых операторов.

Стандартные операторы запроса являются методами, формирующими шаблон LINQ. Большинство этих методов действует в последовательностях, которые представляет собой объект, тип которого реализует интерфейс IEnumerable<T> или интерфейс

IQueryable<T>. Стандартные операторы запросов предоставляют возможности запроса, включая фильтрацию, проекции, статистическую обработку, сортировку и многое другое.

Существуют два набора стандартных операторов запросов LINQ: один, работающий с объектами типа IEnumerable<T>, и другой, работающий с объектами типа IQueryable<T>. Методы, составляющие каждый набор, являются статическими членами классов Enumerable и Queryable соответственно. Они определяются как методы расширения типа, с которым они работают. Это значит, что их можно вызывать либо с помощью синтаксиса статического метода, либо с помощью синтаксиса метода экземпляра.

Кроме того, несколько методов стандартных операторов запросов работают с типами, отличными от основанных на IEnumerable<T> или IQueryable<T>. Тип Enumerable определяет два таких метода, которые оба работают с объектами типа IEnumerable. Эти методы – Cast<TResult>(IEnumerable) и OfType<TResult> (IEnumerable) – позволяют использовать в шаблоне LINQ запросы к непараметризованным и неуниверсальным коллекциям. Для этого они создают строго типизированную коллекцию объектов. Класс Queryable определяет два схожих метода, – Cast<TResult> (IQueryable) и OfType<TResult> (IQueryable) – которые работают с объектами типа Queryable.

Стандартные операторы запросов отличаются по времени выполнения в зависимости от того, возвращают они одноэлементное значение или последовательность значений. Методы, возвращающие одноэлементное значение (например, Average и Sum), выполняются немедленно. Методы, возвращающие последовательность элементов, откладывают выполнение запроса и возвращают перечисляемый объект.

При использовании методов, которые работают с коллекциями в памяти (это методы, которые расширяют IEnumerable<T>), воз-

вращаемый перечисляемый объект захватывает аргументы, переданные в метод. При перечислении объекта используется логика оператора запроса, и возвращаются результаты запроса.

Напротив, методы, расширяющие IQueryable<T>, не реализуют какое-либо поведение запроса, а строят дерево выражения, которое представляет выполняемый запрос. Обработка запроса выполняется исходным объектом IQueryable<T>.

Сортировка

Далее перечислены методы стандартных операторов запроса, которые выполняют сортировку данных:

– OrderBy – по возрастанию;
– OrderByDescending – по убыванию;
– ThenBy – дополнительная сортировка;
– ThenByDescending – дополнительная сортировка по убыванию;
– Reverse – изменение порядка.

Фильтрация

Фильтрацией называется операция ограничения результирующего множества; оно должно содержать только те элементы, которые удовлетворяют указанному условию.

Методы стандартных операторов запросов, которые выполняют выборку, перечислены ниже:

– OfType – выбирает значения в зависимости от возможности приведения их к указанному типу;
– Where – выбирает значения, основанные на функции предиката.

Квантификаторы

Квантификаторы возвращают значение Boolean, которое указывает, удовлетворяют ли условию некоторые или все элементы в последовательности.

Методы стандартных операторов запросов, которые выполняют операции квантификатора, перечислены ниже:

- All – определяет, все ли элементы последовательности удовлетворяют условию;
- Any – определяет, удовлетворяют ли некоторые элементы последовательности условию;
- Contains – проверяет, содержит ли последовательность указанный элемент.

Проецирование

Проекцией называют операцию преобразования объекта в новую форму, которая часто состоит только тех его свойств, которые будут использоваться впоследствии. С помощью проекции можно создать новый тип, который формируется из каждого объекта. Можно проецировать свойство и выполнять над ним математические функции. Также можно проецировать исходный объект, не изменяя его.

Методы стандартных операторов запросов, которые выполняют проецирование, перечислены ниже:

- Select – проецирует значения, основанные на функции преобразования;
- SelectMany – проецирует последовательности значений, основанных на функции преобразования, а затем выравнивает их в одну последовательность.

Преобразование типов

Методы преобразования изменяют тип входных объектов. Операции преобразования в запросах LINQ полезны для различных приложений. Ниже приведены некоторые примеры:

- метод Enumerable.AsEnumerable<TSource> может использоваться для скрытия в типе пользовательской реализации стандартного оператора запроса;

– метод Enumerable.OfType<TResult> может использоваться для поддержки запросов LINQ к непараметризованным коллекциям;
– методы Enumerable.ToArray<TSource>, Enumerable.ToDictionary, Enumerable.ToList<TSource> и Enumerable.ToLookup можно использовать для принудительного немедленного выполнения запроса вместо откладывания выполнения до перечисления запроса;
– Cast – приводит элементы коллекции к указанному типу;
– ToArray – преобразует коллекцию в массив;
– ToList – преобразует коллекцию в List<T>.

Получение элементов

Операции с элементами возвращают один определенный элемент из последовательности.

Методы стандартных операторов запросов, которые выполняют операции с элементами, перечислены ниже:
– ElementAt – возвращает элемент коллекции с указанным индексом;
– ElementAtOrDefault – возвращает элемент коллекции с указанным индексом или значение по умолчанию, если индекс выходит за пределы допустимого диапазона;
– First – возвращает первый элемент коллекции или первый элемент, удовлетворяющий условию;
– FirstOrDefault – возвращает первый элемент коллекции или первый элемент, удовлетворяющий условию. Если такой элемент не существует, возвращает значение по умолчанию;
– Last – возвращает последний элемент коллекции или последний элемент, удовлетворяющий условию;
– LastOrDefault – возвращает последний элемент коллекции или последний элемент, удовлетворяющий условию.

Если такой элемент не существует, возвращает значение по умолчанию;

- Single – возвращает единственный элемент коллекции или единственный элемент, удовлетворяющий условию;
- SingleOrDefault – возвращает единственный элемент коллекции или единственный элемент, удовлетворяющий условию. Если такой элемент отсутствует или коллекция содержит не один такой элемент, возвращает значение по умолчанию.

Разделение

Секционированием в LINQ называют операцию разделения входной последовательности на два раздела без изменения порядка элементов, а затем возвращения одного из разделов.

Методы стандартных операторов запросов, которые выполняют секционирование, перечислены ниже:

- Skip – пропускает элементы до указанной позиции в последовательности;
- SkipWhile – пропускает элементы, пока элемент не удовлетворит условию функции предиката;
- Take – возвращает элементы на указанную позицию в последовательности;
- TakeWhile – принимает элементы, пока элемент не удовлетворит условию функции предиката;
- Distinct – Удаляет повторяющиеся значения из коллекции.

Статистика

Статистическая операция вычисляет одно значение из коллекции значений.

Методы стандартных операторов запросов, которые выполняют статистические операции:

– Average – вычисляет среднее значение коллекции значений;
– Count – подсчитывает число элементов в коллекции (при необходимости только те элементы, которые удовлетворяют функции предиката);
– Max – определяет максимальное значение в коллекции;
– Min – определяет минимальное значение в коллекции;
– Sum – вычисляет сумму значений в коллекции.

Приведем **пример решения** задач на языке LINQ, при следующем описании классов объектов предметной области: университет (название, список групп), группа (номер, список студентов), студент (фамилия, имя, отчество, средний балл). Требуется завершить преобразование LINQ методом ToArray, ToList.

Запросы:

1. Отсортировать группы по номеру группы.
2. Отсортировать студентов в группе по имени, затем по среднему баллу.
3. Получить всех студентов в группе, у которых средний балл выше заданного.
4. Получить все группы, в которых число студентов выше заданного.
5. Узнать, если ли в группе хоть один студент с именем Иван.
6. Узнать, может всех студентов в группе зовут Марина.
7. Получить список имен всех студентов в группе.
8. Получить список имен всех студентов в группе без повторений.
9. Получить первого студента в группе с именем Иван.
10. Получить последнего студента в группе с именем Марина.
11. Получить единственного студента в группе с именем Арнольд.

12. Взять первых 5 студентов из группы.

13. Пропустить первых 3 студентов и следующих 4.

14. Пропустить всех студентов, пока не встретится имя Марина, и взять всех остальных.

15. Найти максимальный балл студента в группе.

16. Найти число студентов, у которых средний балл выше заданного.

Решение:

```
public class University
   {
      private readonly List<Group> _groups;

      public string Name { get; }

      public List<Group> Groups =>
         new List<Group>(_groups);

      public University(string name, List<Group> groups)
      {
         Name = name;
         _groups = groups;
      }
   }


public class Group
   {
      public int Number { get; }

      private readonly List<Student> _students;

      public List<Student> Students =>
         new List<Student>(_students);

      public Group(int number, List<Student> students)
      {
         Number = number;
```

```
            _students = students;
        }
        public override string ToString()
        {
            return "Group " + Number;
        }
    }


    public class Student
    {
        public FullName FullName { get; }

        public double Mark { get; }

        public Student(FullName fullName, double mark)
        {
            if (mark < 2 || mark > 5) {
                throw new ArgumentOutOfRangeException(nameof(mark), "Stu-
dent mark must be bound in range 2-5.");
            }

            FullName = fullName;
            Mark = mark;
        }

        public override string ToString()
        {
            return FullName.ToString();
        }
    }

    public class FullName
    {
        public string Firstname { get; }

        public string Lastanme { get; }

        public string Patronymic { get; }

        public FullName(string firstname, string lastanme, string patronymic)
```

```
            {
                Firstname = firstname;
                Lastanme = lastanme;
                Patronymic = patronymic;
            }

            public override string ToString()
            {
                return Firstname + (Patronymic != null ? (" " + Patronymic + " ")
: " ") +  Lastanme;
            }
        }

    internal class Program
        {
            public static void Main(string[] args)
            {
                Student ivanovIvan = new Student(new FullName("Ivan", "Ivanov",
"Fedorovich"), 4.2);
                Student sidorovAnton = new Student(new FullName("Anton", "Si-
dorov", "Alexeevich"), 3.4);
                Student voronovArtur = new Student(new FullName("Artur", "Vo-
ronov", "Michaylovich"), 5.0);
                Student     schwarzeneggerArnold     =     new     Student(new
FullName("Arnold", "Schwarzenegger", null), 4.8);
                Student   kravcovIvan   =   new   Student(new   FullName("Ivan",
"Kravcov", "Sergeevich"), 4.1);
                Student semonovaMarina = new Student(new FullName("Marina",
"Semenova", "Dmitrievna"), 4.0);
                Student   topolevaAnna   =   new   Student(new   FullName("Anna",
"Topoleva", "Gennadevna"), 4.6);
                Student     zhmishenkoValeriy     =     new     Student(new
FullName("Valeriy", "Zhmishenko", "Albertovich"), 5.0);
                Student   suhachovDenis   =   new   Student(new   FullName("Denis",
"Suhachov", "Victorovich"), 2.0);

                Group group1 = new Group(1, new List<Student>() {
                    ivanovIvan,
                    sidorovAnton,
                    voronovArtur,
                    schwarzeneggerArnold,
```

```
                    kravcovIvan,
                    semonovaMarina,
                    topolevaAnna,
                    zhmishenkoValeriy,
                    suhachovDenis
                });

                Student alexeevaDaria = new Student(new FullName("Daria", "Aelxeeva", "Sergeevna"), 4.0);
                Student nosovGerman = new Student(new FullName("German", "Nosov", "Georgievich"), 4.2);
                Student panovaEkaterina = new Student(new FullName("Ekaterina", "Panova", "Vladimirovna"), 3.9);

                Group group2 = new Group(2, new List<Student>() {
                    alexeevaDaria,
                    nosovGerman,
                    panovaEkaterina
                });

                Group group3 = new Group(3, new List<Student>());

                University university = new University("Fake university", new List<Group>() {
                    group1,
                    group2,
                    group3
                });

                Task1(university);
                Task2(group1);
                Task3(group1, 4);
                Task4(university, 5);
                Task5(group1);
                Task6(group1);
                Task7(group1);
                Task8(group1);
                Task9(group1);
                Task10(group1);
                Task11(group1);
                Task12(group1);
```

```
    Task13(group1);
    Task14(group1);
    Task15(group1);
    Task16(group1, 4);
}

private static void Task1(University university)
{
    var groups = from universityGroup in university.Groups
        orderby universityGroup.Number
        select universityGroup;

    Console.WriteLine("Task 1:  " + string.Join("\n\t ", groups));
}

private static void Task2(Group @group)
{
    var students = from student in @group.Students
        orderby student.FullName.ToString(), student.Mark
        select student;

    Console.WriteLine("Task 2:  " + string.Join("\n\t ", students));
}

private static void Task3(Group group1, double markThreshold)
{
    var students = from student in group1.Students
        where student.Mark > markThreshold
        select student;

    Console.WriteLine("Task 3:  " + string.Join("\n\t ", students));
}

private static void Task4(University university, int studentThreshold)
{
    var groups = from @group in university.Groups
        where @group.Students.Count > studentThreshold
        select @group;

    Console.WriteLine("Task 4:  " + string.Join("\n\t ", groups));
}
```

```
private static void Task5(Group @group)
{
    var students = from student in @group.Students
        where student.FullName.Firstname.Equals("Ivan")
        select student;

    Console.WriteLine("Task 5:  " + students.Any());
}

private static void Task6(Group @group)
{
    Console.WriteLine("Task 6:  " + @group.Students.All(student =>
student.FullName.Firstname.Equals("Marina")));
}

private static void Task7(Group @group)
{
    var students = from student in @group.Students
        select student.FullName.Firstname;

    Console.WriteLine("Task 7:  " + string.Join("\n\t ", students));
}

private static void Task8(Group @group)
{
    var students = from student in @group.Students
        group student.FullName.Firstname by stu-
dent.FullName.Firstname into f
        select f.Key;

    Console.WriteLine("Task 8:  " + string.Join("\n\t ", students));
}

private static void Task9(Group @group)
{
    var ivan = (from student in @group.Students
        where student.FullName.Firstname.Equals("Ivan")
        select student).First();

    Console.WriteLine("Task 9:  " + ivan);
}
```

```
private static void Task10(Group @group)
{
   var marina = (from student in @group.Students
      where student.FullName.Firstname.Equals("Marina")
      select student).Last();

   Console.WriteLine("Task 10: " + marina);
}

private static void Task11(Group @group)
{
   var arnold = (from student in @group.Students
      where student.FullName.Firstname.Equals("Arnold")
      select student).FirstOrDefault();

   Console.WriteLine("Task 11: " + (arnold != null ? ar-
nold.ToString() : "There is no student with name 'Arnold'"));
}

private static void Task12(Group @group)
{
   var students = (from student in @group.Students
      select student).Take(5).ToList();

   Console.WriteLine("Task 12: " + string.Join("\n\t ", students));
}

private static void Task13(Group @group)
{
   var students = (from student in @group.Students
      select student).Skip(3).Take(4).ToList();

   Console.WriteLine("Task 13: " + string.Join("\n\t ", students));
}

private static void Task14(Group @group)
{
   var students = (from student in @group.Students
      select student).SkipWhile(each =>
!each.FullName.Firstname.Equals("Marina"));
```

```
        Console.WriteLine("Task 14: " + string.Join("\n\t ", students));
    }

    private static void Task15(Group @group)
    {
        var max = (from student in @group.Students
            select student.Mark).Max();

        Console.WriteLine("Task 15: " + max);
    }

    private static void Task16(Group @group, float markThreshold)
    {
        var count = (from student in @group.Students
            where student.Mark > markThreshold
            select student).Count();

        Console.WriteLine("Task 16: " + count);
    }
}
```

2.5 Эффективность алгоритмов и оптимизация кода

Рассматриваются вопросы эффективности алгоритмов и способы выполнения их оптимизации (по процессорному времени и по используемой памяти). Приводятся примеры оптимизации программы и ее рефакторинга.

СПО должно удовлетворять следующим требованиям:

- прозрачность работы;
- гарантированная надежность выполнения в соответствии со спецификациями;
- максимальная скорость выполнения;
- минимальные затраты на хранение машинных кодов;
- поддержка стандартных средств связи с прикладными программами.

Эффективность алгоритма – это свойство алгоритма, которое связано с вычислительными ресурсами, используемыми алгоритмом. Алгоритм должен быть проанализирован с целью определения необходимых алгоритму ресурсов.

Рассмотрим процесс оптимизации СПО на примере модификации программного кода:

1) Рассмотри метод CsvWriter для записи данных в файл:

```
public void CsvWriter(string way)
{
   try
   {
      StreamWriter sw = new StreamWriter(way, false, Encoding.Default);
      for (int i = 0; i < dataGridView1.ColumnCount; i++)
      {
         sw.Write(dataGridView1.Columns[i].Name);
         if (i < dataGridView1.ColumnCount - 1)
            sw.Write(',');
      }
      sw.WriteLine();

      for (int i = 0; i < dataGridView1.RowCount; i++)
      {
         for (int j = 0; j < dataGridView1.ColumnCount; j++)
         {
            sw.Write(dataGridView1[j, i].Value);
            if (j < dataGridView1.ColumnCount - 1)
               sw.Write(',');
         }
         sw.WriteLine();
      }

      sw.Close();
   }
   catch (Exception e)
   {
      MessageBox.Show("Ошибка в блоке записи CSV файла");
      LogException.WriteLog(e, "Ошибка в блоке записи CSV файла");
```

```
    }
}
```

Выражение для получения числа столбцов запишем в отдельную перемененную.

```
public void CsvWriter(string way)
{
    try
    {
        StreamWriter sw = new StreamWriter(way, false, Encoding.Default);
        int cc = dataGridView1.ColumnCount;
        for (int i = 0; i < cc; i++)
        {
            sw.Write(dataGridView1.Columns[i].Name);
            if (i < cc - 1)
                sw.Write(',');
        }
        sw.WriteLine();

        for (int i = 0; i < dataGridView1.RowCount; i++)
        {
            for (int j = 0; j < cc; j++)
            {
                sw.Write(dataGridView1[j, i].Value);
                if (j < cc - 1)
                    sw.Write(',');
            }
            sw.WriteLine();
        }

        sw.Close();
    }
    catch (Exception e)
    {
        MessageBox.Show("Ошибка в блоке записи CSV файла");
        LogException.WriteLog(e, "Ошибка в блоке записи CSV файла");
    }
}
```

2) В inputB_Click удалим повторное создание экземпляра класса FileInfo.

Код до:

```
private void inputB_Click(object sender, EventArgs e)
{
   try
   {
      MajorForm major = this.Owner as MajorForm;
      major.listAdd(inputB.Text);
      if (dataGridView1.Enabled == false)
      {
         MessageBox.Show("Загрузите .CSV файл");
      }
      else
      {
         if (openFileDialog2.ShowDialog() == DialogResult.OK)
         {
            FileInfo fileEx = new FileInfo(openFileDialog2.FileName);
            string fe = fileEx.Extension;
            if (fe == ".exe")
            {
               string a, b, c = null;
               FileInfo fileIn = new FileInfo(openFileDialog2.FileName);
               FileVersionInfo fileversion = FileVersionInfo.GetVersionInfo(openFileDialog2.FileName);
               a = fileIn.Name;
               b = fileversion.FileVersion;
               c = fileIn.LastWriteTime.ToShortDateString();
               list.Add(a + ',' + b + ',' + c);
               UpdateTable();
               UpdateList();
            }
            else throw new MyException("Выберите файл с расширением ", " .EXE");
         }
      }
   }
   catch (MyException ex)
```

```
            {
               MessageBox.Show(ex.Message + “-“ + ex.FileEx);
               LogException.WriteLog(ex, ex.Message + “-“ + ex.FileEx);
            }
            catch (Exception ex)
            {
               MessageBox.Show(“Ошибка в блоке загрзуки EXE файла
(диалоговое окно)”);
               LogException.WriteLog(ex, “Ошибка в блоке загрзуки EXE
файла (диалоговое окно)”);
            }
        }
```

Код после:

```
private void inputB_Click(object sender, EventArgs e)
        {
            try
            {
               MajorForm major = this.Owner as MajorForm;
               major.listAdd(inputB.Text);
               if (dataGridView1.Enabled == false)
               {
                  MessageBox.Show("Загрузите .CSV файл");
               }
               else
               {
                  if (openFileDialog2.ShowDialog() == DialogResult.OK)
                  {
                     FileInfo fileEx = new FileInfo(openFileDialog2.FileName);
                     string fe = fileEx.Extension;
                     if (fe == ".exe")
                     {
                        string a, b, c = null;
                        FileVersionInfo fileversion = FileVersionIn-
fo.GetVersionInfo(openFileDialog2.FileName);
                        a = fileEx.Name;
                        b = fileversion.FileVersion;
                        c = fileEx.LastWriteTime.ToShortDateString();
                        list.Add(a + ',' + b + ',' + c);
                        UpdateTable();
                        UpdateList();
                     }
```

```
                    else throw new MyException("Выберите файл с расшире-
нием ", " .EXE");
                  }
               }
            }
            catch (MyException ex)
            {
               MessageBox.Show(ex.Message + "-" + ex.FileEx);
               LogException.WriteLog(ex, ex.Message + "-" + ex.FileEx);
            }
            catch (Exception ex)
            {
               MessageBox.Show("Ошибка в блоке загрзуки EXE файла
(диалоговое окно)");
               LogException.WriteLog(ex, "Ошибка в блоке загрзуки EXE
файла (диалоговое окно)");
            }
      }
```

3) В deleteB_Click удалим лишние логические вычисления. Вместо двух if сделаем один.

Код до:

```
private void deleteB_Click(object sender, EventArgs e)
      {
         try
         {
            MajorForm major = this.Owner as MajorForm;
            major.listAdd(deleteB.Text);

            if (dataGridView1.Enabled == false)
            {
               MessageBox.Show("Загрузите .CSV файл");
            }
            else
            {
               if (dataGridView1.CurrentRow == null)
               {
                  MessageBox.Show("Выделите строку для удаления");
               }
```

```
                else
                {
                    list.DeleteIndex(dataGridView1.CurrentRow.Index);
                    UpdateTable();
                    UpdateList();
                }
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Ошибка в блоке удаления записи");
            LogException.WriteLog(ex, "Ошибка в блоке удаления запи-
си");
        }
}
```

Код после:

```
private void deleteB_Click(object sender, EventArgs e)
    {
        try
        {
            MajorForm major = this.Owner as MajorForm;
            major.listAdd(deleteB.Text);

                if (dataGridView1.CurrentRow == null)
                {
                    MessageBox.Show("Загрузите .CSV файл или выделите
строку для удаления");
                }
                else
                {
                    list.DeleteIndex(dataGridView1.CurrentRow.Index);
                    UpdateTable();
                    UpdateList();
                }

        }
        catch (Exception ex)
        {
            MessageBox.Show("Ошибка в блоке удаления записи");
```

LogException.WriteLog(ex, "Ошибка в блоке удаления записи");
}
}

Оценка эффективности разработанного СПО и проведенной оптимизации проведена в профайлере EQATEC Profiler.

Оценка эффективности функционирования СПО до оптимизации приведена на рисунке 4 в колонке «Old Avg(Full)». Оценка эффективности функционирования СПО после оптимизации приведена на рисунке 4 в колонке «New Avg(Full)». Как видно из рисунка, общее время выполнения программы сократилось на 1644 мс, что говорит об эффективности принятых изменений.

Confidence	Speedup	Diff Avg(full)	Old Avg(full)	New Avg(full)	Diff Calls	Old Calls	New Calls	Method name
★★★★	1.06 x	-1 644	28 445	26 801	0	1	1	SP_Ganeev_11.Program.Main
★★★★	1.08 x	-176	2 487	2 311	0	1	1	SP_Ganeev_11.Form2.saveB_Click(System.Object,System.EventArgs)
★★★★	1.09 x	-149	1 895	1 746	0	1	1	SP_Ganeev_11.Form2.inputB_Click(System.Object,System.EventArgs)
★★★★	1.04 x	-79	2 205	2 126	0	1	1	SP_Ganeev_11.Form2.downloadB_Click(System.Object,System.EventArgs)
★★★	1.57 x	-4.0	11	7.0	0	1	1	SP_Ganeev_11.Form1.Dispose(System.Boolean)
★★½	1.50 x	-2.0	6.0	4.0	0	1	1	SP_Ganeev_11.MajorForm.Dispose(System.Boolean)
★★½	1.25 x	-1.0	5.0	4.0	0	1	1	SP_Ganeev_11.Form1.ctor
★★	2.00 x	-1.0	2.0	1.0	0	1	1	SP_Ganeev_11.Form2.deleteB_Click(System.Object,System.EventArgs)
★★★	1.04 x	-1.0	24	23	0	1	1	SP_Ganeev_11.Form2.CsvWriter(System.String)
★★	1.00 x	0	1.0	1.0	0	1	1	SP_Ganeev_11.Form2.UpdateTable
★★★	1.00 x	0	20	20	0	1	1	SP_Ganeev_11.Form2.Dispose(System.Boolean)
★★	1.00 x	0	2.0	2.0	0	1	1	SP_Ganeev_11.MajorForm.ctor
★★	1.00 x	0	1.0	1.0	0	2	2	SP_Ganeev_11.Form2.CsvReader(System.String)
★★½	1.00 x	0	6.0	6.0	0	1	1	SP_Ganeev_11.Form3.ctor
★★★	1.00 x	0	12	12	0	1	1	SP_Ganeev_11.Form3.button1_Click(System.Object,System.EventArgs)
★★	1.00 x	0	2.0	2.0	0	1	1	SP_Ganeev_11.Form2.editB_Click(System.Object,System.EventArgs)
★★★	1.00 x	0	21	21	0	1	1	SP_Ganeev_11.MajorForm.button1_Click(System.Object,System.EventArgs)
★★★	1.00 x	0	17	17	0	1	1	SP_Ganeev_11.MajorForm.button3_Click(System.Object,System.EventArgs)
★★★	1.00 x	0	14	14	0	1	1	SP_Ganeev_11.MajorForm.InitializeComponent

Рисунок 4 – Оценка эффективности функционирования СПО

2.6 Тестирование, отладка, структурная обработка исключений

Рассматриваются вопросы организации тестирования программ, в том числе процесс отладки программы. Даются сведения об автоматизированном юнит-тестировании программ. Даются сведения об интеграционном тестировании программ.

Тестирование проводится не только на той стадии разработки программ, которая специально для этого предназначена, но и на

предшествующих стадиях –при автономной отладке программ, еще до объединения их в единый программный комплекс. Такое тестирование называется *модульным*. Его обычно проводят сами разработчики, которые проверяют точное соответствие программы выданной им спецификации.

Интеграционное тестирование призвано проверить все аспекты работы программы от правильности взаимодействия внутренних программных компонентов до правильности взаимодействия программного комплекса с его пользователями.

Во время *пользовательского тестирования* результаты работы программы проверяются с прикладной точки зрения.

Нагрузочное тестирование дает возможность проверить безопасную и эффективную работу созданной программы в нормальном и пиковом режимах ее использования. Функциональность на этом этапе проверяется только в смысле ее влияния на важнейшие технические параметры программы, например, на время реакции системы на запрос пользователя.

Важной в тестировании является возможность проведения *регрессионного тестирования*. Регрессионные тесты, повторяемые после каждого исправления программы, позволяют убедиться, что функциональность программы, не связанная с внесенным исправлением, не затронута этим исправлением и не утрачена из-за него.

Модульное (unit) тестирование – практические методы, применяемые при разработке программного обеспечения, повышающие надежность ПО, способствующие упрощению внутренней архитектуры приложения. Создание модульных тестов позволяет сократить число потенциальных ошибок ПО, что уменьшает стоимость сопровождения.

Примеры инструментальных средств

Средства тестирования: стандартные средства IDE, NUnit, Moq.

Средства логирования: log4net.

Системы отслеживания ошибок: Redmine, YouTrack, Bugzilla.

Непрерывная интеграция: TeamCity, Team Foundation Server, Travis CI, CruiseControl.

Покрытие тестами: dotCover.

Службы хранения сообщений: Graylog2.

2.7 Прочие вопросы

В настоящем учебном пособии не рассмотрены вопросы создания, документирования, распространения и защиты динамически-подключаемых библиотек (DLL). С материалом по указанным вопросам можно ознакомиться в [21, 25, 30].

Также, вне фокуса учебного пособия осталось метапрограммирование и, в частности, рефлексия, подробно рассмотренные, например, в [25].

Кроме этого, обучающимся рекомендуется ознакомиться с вопросами разработки драйверов и сходств и различий в драйверах, подготовленных для операционных систем Windows и Linux [26, 28].

3 ЛАБОРАТОРНЫЙ ПРАКТИКУМ

3.1 Общие требования к разрабатываемому программному обеспечению

Системное ПО (СПО) должно иметь графический пользовательский интерфейс, не нарушающий принятые подходы к проектированию интерфейсов.

СПО должно обеспечивать корректную работу с вводимыми данными, обеспечивать обработку исключительных ситуаций, в случае их возникновения.

СПО должно иметь возможность сохранения в файл и загрузки из файла введенных пользователем данных. Вводимые данные и формат файла определяются индивидуальным вариантом. Пользователь должен иметь возможность добавить, удалить, отредактировать запись данных. Записи данных представляются в табличном виде.

СПО должно обеспечить разбор и анализ вводимой с клавиатуры конструкции языка, определенной индивидуальным вариантом, и визуализацию результата.

СПО должно обеспечить выполнение низкоуровневого кода с передачей параметров, введённых пользователем, и визуализацию результата.

В процессе работы СПО должно формировать записи текущей работы («лог сообщений»), отображаемые на экране.

СПО в минимальной реализации должно состоять из одного исполняемого файла (exe) и двух динамически подключаемых библиотек (dll). Одна из библиотек содержит вызов низкоуровневых функций, соответствующих индивидуальному варианту, другая – бизнес-логику приложения. Исполняемый файл содержит элементы графического пользовательского интерфейса, программный код для

организации человеко-машинного взаимодействия и программный код, обращающийся к динамически подключаемым библиотекам.

Выбор языков программирования, используемых в ходе выполнения лабораторных работ, остается за студентом. Выбранные языки программирования должны обеспечить возможность создания СПО по указанным требованиям. Рекомендуется использовать язык программирования C# с низкоуровневыми вставками на языке MSIL(CIL) с использованием Emit. В случае необходимости, допустимо использовать unsafe / unchecked код и механизм DllImport.

3.2 Порядок выполнения лабораторных работ

Цель курса лабораторных работ – получение практических навыков проектирования, разработки, отладки, тестирования и документирования системного программного обеспечения (СПО).

Курс состоит из 9 лабораторных работ, выполняемых по выбранному индивидуальному варианту задания. Лабораторная работа №6 носит необязательный характер и направлена на развитие навыков использования современных технологий доступа к данным.

Лабораторная работа выполняется на компьютере с установленным программным обеспечением:

– текстовый процессор;
– среда проектирования программного обеспечения;
– среда разработки программного обеспечения на языке высокого уровня.

Результаты выполнения лабораторной работы предъявляются преподавателю и защищаются во время аудиторных занятий. Защита проходит в форме собеседования, форма отчетности – «зачтено/неудовлетворительно». Лабораторная работа может потребовать доработки при наличии существенных замечаний по содержанию материала, несоответствии индивидуального варианта подготов-

ленной работе, несоответствии оформления работы требованиям СТО 02068410-004-2018 «Общие требования к учебным текстовым документам» [29].

3.3 Индивидуальные варианты

Лабораторные работы выполняются студентом индивидуально или группой студентов (2-3 человека). В случае выполнения лабораторной работы группой студентов, необходимо реализовать все индивидуальные варианты студентов, объединенных в группу.

Индивидуальный вариант студента определяется двумя последними цифрами номера зачетной книжки студента. Если две последние цифры номера книжки превышают число вариантов, то номер варианта определяется путем сложения двух последних цифр номера. Индивидуальные варианты приведены в таблице 4.

Примеры:

– 13 (такой вариант есть) = 13 вариант;

– 26 (такого варианта нет, складываем) = 2+6= 8 вариант;

– 99 (такого варианта нет, складываем) = 9+9 = 18 (такого варианта нет, значит снова складываем) = 1 + 8 = 9 вариант.

По усмотрению преподавателя, проводящего курс лабораторных работ, возможен другой порядок назначения варианта задания.

Таблица 4 – Индивидуальные варианты

№	Реализуемая конструкция языка (синтаксис C#)	Формат файла	Вводимые данные	Низкоуровневая функция
1.	Итеративный цикл for(<инициализация>; <условие>; <итератор>) { <тело цикла> } Посчитать, сколько раз выполнится цикл	XML	Запись о сервере: адрес сервера, порт, протокол (TCP или UDP). Для добавления новой записи вывести окно ввода	Сложить два целых числа со знаком с проверкой переполнения

Продолжение табл. 4

№	Реализуемая конструкция языка (синтаксис C#)	Формат файла	Вводимые данные	Низкоуровневая функция
2.	Цикл с предусловием while (<условие>) { <тело цикла> } Определить, выполнится ли цикл хотя бы раз	JSON	Запись о файле: путь к файлу, размер файла в килобайтах, дата создания файла. Для добавления новой записи необходимо указать файл на компьютере	Сложить два целых числа без знака с проверкой переполнения
3.	Цикл с постусловием do {<тело цикла>} while (<условие>) Определить, выполнится ли цикл больше одного раза	CSV	Запись о файле: путь к файлу, размер файла в мегабайтах, дата последнего редактирования файла. Для добавления новой записи необходимо указать файл на компьютере	Вычислить побитовое И двух значений
4.	Оператор условного перехода if (<условие>) { <действие 1> } [else {<действие 2>}] Определить, какое из условий выполнится	Plain text	Запись о файле (*.dll): имя файла, версия файла, дата последнего редактирования файла. Для добавления новой записи необходимо указать файл на компьютере	Сравнить два значения на равенство
5.	Цикл-перебор foreach (<элемент> in <массив>) { <тело цикла> } Посчитать, сколько раз выполнится цикл	Binary	Запись о файле (*.exe): имя файла, версия файла, дата создания файла. Для добавления новой записи необходимо указать файл на компьютере	Сравнить два значения с условием «строго больше»

Продолжение табл. 4

№	Реализуемая конструкция языка (синтаксис C#)	Формат файла	Вводимые данные	Низкоуровневая функция
6.	Итеративный цикл for(<инициализация>; <условие>; <итератор>) { <тело цикла> } Посчитать, сколько раз выполнится цикл	JSON	Запись о доступе: логин, хэш-код пароля, email. Для добавления новой записи вывести окно ввода	Сравнить два значения с условием «строго меньше»
7.	Цикл с предусловием while (<условие>) { <тело цикла> } Определить, выполнится ли цикл хотя бы раз	CSV	Запись о ресурсе: адрес ресурса в сети Интернет, режим доступа (свободный, закрытый), дата доступа. Для добавления новой записи вывести окно ввода	Разделить одно значение на другое
8.	Цикл с постусловием do {<тело цикла>} while (<условие>) Определить, выполнится ли цикл больше одного раза	Plain text	Запись о ресурсе: адрес ресурса в сети Интернет, режим доступа (свободный, закрытый), дата доступа. Для добавления новой записи вывести окно ввода	Разделить одно целочисленное значение без знака на другое
9.	Оператор условного перехода if (<условие>) { <действие 1> } [else {<действие 2>}] Определить, какое из условий выполнится	Binary	Запись о доступе: логин, хэш-код пароля, email. Для добавления новой записи вывести окно ввода	Умножить два целочисленных значения с проверкой переполнения
10.	Цикл-перебор foreach (<элемент> in <массив>) { <тело цикла> } Посчитать, сколько раз выполнится цикл	XML	Запись о файле (*.dll): имя файла, версия файла, дата последнего редактирования файла	Умножить два целочисленных значения без знака с проверкой переполнения

Продолжение табл. 4

№	Реализуемая конструкция языка (синтаксис C#)	Формат файла	Вводимые данные	Низкоуровневая функция
			Для добавления новой записи необходимо указать файл на компьютере	
11.	Итеративный цикл for(<инициализация>; <условие>; <итератор>) { <тело цикла> } Посчитать, сколько раз выполнится цикл	CSV	Запись о файле (*.exe): имя файла, версия файла, дата создания файла. Для добавления новой записи необходимо указать файл на компьютере	Вычислить побитовое ИЛИ двух значений
12.	Цикл с предусловием while (<условие>) { <тело цикла> } Определить, выполнится ли цикл хотя бы раз	Plain text	Запись о сервере: адрес сервера, порт, протокол (TCP или UDP). Для добавления новой записи вывести окно ввода	Вычислить побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ двух значений
13.	Цикл с постусловием do {<тело цикла>} while (<условие>) Определить, выполнится ли цикл больше одного раза	Binary	Запись о файле: путь к файлу, размер файла в килобайтах, дата создания файла. Для добавления новой записи необходимо указать файл на компьютере	Вычислить побитовое дополнение (НЕ)
14.	Оператор условного перехода if (<условие>) { <действие 1> } [else {<действие 2>}] Определить, какое из условий выполнится	XML	Запись о файле: путь к файлу, размер файла в мегабайтах, дата последнего редактирования файла.	Разделить одно значение на другое

Окончание табл. 4

№	Реализуемая конструкция языка (синтаксис C#)	Формат файла	Вводимые данные	Низкоуровневая функция
			Для добавления новой записи необходимо указать файл на компьютере	
15.	Цикл-перебор foreach (<элемент> in <массив>) { <тело цикла> } Посчитать, сколько раз выполнится цикл.	JSON	Запись о файле (*.dll): имя файла, версия файла, дата последнего редактирования файла. Для добавления новой записи необходимо указать файл на компьютере	Разделить одно значение без знака на другое значение без знака

3.4 Анализ требований к системному программному обеспечению

Основными этапами разработки ПО являются:

– подготовка;
– проектирование;
– написание исходных текстов программ;
– тестирование и отладка программ;
– испытания и сдача программ;
– сопровождение программ.

Одним из этапов разработки ПО является подготовка. Этап подготовки включает в себя сбор и анализ требований к разрабатываемому ПО, планирование этапов выполняемых работ и сроков их исполнения, необходимых ресурсах и стоимости.

В рамках подготовки к разработке СПО необходимо оформить задание к циклу лабораторных работ. Задание оформляется на

бланке, и содержит основные сведения об исполнителях работы, задании и план-график выполнения. Задание должно быть подписано исполнителями и руководителем работ.

На основе анализа требований к разрабатываемому ПО разрабатывается техническое задание. Техническое задание к разрабатываемому СПО оформляется в виде Приложения к заданию и представляет собой перечень требований к разрабатываемому СПО. Разработка технического задания выполняется на основе ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению». Приложение включает следующие сведения:

1 Цель цикла лабораторных работ

2 Требования к системе в целом

В данном разделе приводятся нефункциональные, общие требования, которые описывают создаваемое ПО с разных сторон.

3 Требования к функциям, выполняемым разрабатываемым СПО

В данном разделе описываются:

– требования к функциям, выполняемым СПО, которые представляют собой перечень функций или задач, подлежащих автоматизации;

– приводятся требования к форме представления входной информации;

– приводятся требования к качеству реализации функций.

4 Требования к структуре СПО

В разделе приводится общее представление об архитектуре СПО: подсистемы, модули и библиотеки, из которых должно состоять разрабатываемое СПО.

5 Требования к видам обеспечения

5.1 Требования к информационному обеспечению.

Информационное обеспечение представляет собой совокупность форм документов, классификаторов, нормативной базы и реа-

лизованных решений по объемам, размещению и формам существования информации, применяемой в АС при ее функционировании.

В разделе приводятся требования к входящей информации и информации, передающейся от компонента к компоненту системы в неавтоматизированном виде.

5.2 Требования к программному обеспечению

В разделе приводится перечень стороннего ПО, необходимого для разработки СПО.

5.3 Требования к аппаратному обеспечению:

В разделе определяются конкретные характеристики оборудования, на котором должно выполняться разрабатываемо СПО.

5.4 Требования к лингвистическому обеспечению

Требования к лингвистическому обеспечению включает описание требований к языкам программирования и средам разработки СПО, а также к языку интерфейса для взаимодействия с пользователем.

6 Требования к способам организации диалога с пользователем

В разделе должно быть приведено описание необходимого качества взаимодействия человека с машиной и комфортности условий работы пользователя.

7 Требования к быстродействию

8 Требования к документированию:

В разделе приводятся требования к комплекту разрабатываемой документации и требования к ее оформлению.

Лабораторная работа № 1

Анализ требований к системному программному обеспечению

Цель работы – формирование и анализ требований к СПО.

Ход выполнения работы:

1) Изучить требования, предъявляемые к СПО.

2) Выбрать индивидуальный вариант задания.

3) Сформировать перечень требований к СПО, реализующему индивидуальный вариант.

4) Оформить перечень требований к СПО в виде Приложения к заданию.

5) Согласовать и утвердить Задание и Приложение к заданию у преподавателя.

Результаты работы:

Сформированный перечень требований к СПО, оформленный в виде согласованного и утверждённого преподавателем Задания и Приложения к заданию.

3.5 Проектирование системного программного обеспечения

Важным этапом разработки ПО является его проектирование. Процесс проектирования СПО описан в разделе 2.2.

При выполнении лабораторной работы проектирование ПО выполняется на основе разработанного технического задания. В процессе проектирования осуществляется выбор архитектуры системы, проектируются логическая и физическая модели данных, динамическая и статическая модели ПО, выполняется объектная декомпозиция разрабатываемого ПО. Во время проектирования принимаются решения по выбору программного и аппаратного обеспечения, СУБД, обосновывается сделанный выбор.

Кроме этого, разрабатывается структурная схема СПО. Структурной схемой называется схема, отображающая состав и взаимодействие частей разрабатываемого ПО. Структурными компонентами ПО являются, например, подпрограммы, подсистемы, базы данных, библиотеки ресурсов.

Лабораторная работа № 2
Проектирование системного программного обеспечения

Цель работы – проектирование СПО на уровне модулей.

Ход выполнения работы:

1) Изучить порядок проектирования СПО.
2) На основе разработанного ТЗ выполнить проектирование СПО.
3) Разработать структурную схему СПО.
4) Подготовить перечни требований к программному и аппаратному обеспечению для функционирования СПО.
5) Оформить структурную схему СПО и перечни требований в виде документа.

Результаты работы:

Проект СПО, оформленный в виде документа.

3.6 Создание сложной структуры данных

Под структурами данных понимаются способы представления и хранения данных в памяти. При создании структур данных необходимо учитывать следующие параметры:

- вид хранимых данных элементов структуры;
- время хранения структуры данных;
- связи элементов структур данных;
- совокупность возможных действий с данными и структурами данных.

К наиболее часто используемым структурам данных относятся:

1) линейные структуры: массивы, динамические и связанные массивы. Они линейны потому, что остаются в том порядке, в котором их расположили. Массивы быстры при произвольном доступе и имеют относительно неплохую производительность при добавлении

и удалении из конца. Связанный список удобно применять при частом добавлении и удалении из середины.

2) линейные структуры данных с конечными точками: семейство стеков и очередей. У стеков доступ можно получить только к последнему («верхнему») элементу, как для добавления элемента, так и для удаления. Очередь работает по тому же принципу, что и в реальной жизни: удаляются элементы из начала, а добавляются в конец.

3) нелинейные структуры данных: словарь данных, упорядоченное и неупорядоченное множество.

Лабораторная работа № 3
Проектирование структур данных и отношений между ними

Цель работы – разработка структур данных для использования в СПО.

Ход выполнения работы:

1) Изучить методологию проектирования структур данных.

2) Выполнить проектирование и анализ структур данных, предназначенных для использования в СПО.

3) Разработать структуры данных на языке высокого уровня.

4) Провести документирование структур данных с использованием средств языка (документирующие комментарии) – Javadoc, <summary> и т.п.

Результаты работы:

Структуры данных и документация, описанные на языке высокого уровня.

3.7 Разработка системного программного обеспечения с использованием принципов SOLID и паттернов проектирования

Описание принципов SOLID и паттернов проектирования приведено в разделе 2.3 настоящего учебного пособия.

Лабораторная работа № 4
Разработка системного программного обеспечения с использованием принципов SOLID и паттернов проектирования

Цель работы – разработка СПО на языке высокого уровня с использованием современных подходов к разработке.

Ход выполнения работы:

1) Изучить принципы SOLID.
2) Изучить паттерны проектирования.
3) Разработать на языке высокого уровня СПО с графическим интерфейсом, соблюдая принципы SOLID и реализуя один из паттернов отделения логики от представления пользователю (MVC, MVP, MVVM и т.п.).

Результаты работы:

СПО, написанное на языке высокого уровня.

3.8 Вызов ассемблерных функций из языка высокого уровня

Для вызова ассемблерных функций из языка высокого уровня могут быть использованы следующие подходы.

Для вызова низкоуровневой функции на языке Java используется Java Native Interface (JNI) – стандартный механизм для запуска кода под управлением виртуальной машины Java (JVM), который написан на языках C/C++ или Ассемблере и скомпонован в виде динамических библиотек; позволяет не использовать статическое связывание, что даёт возможность вызова функции C/C++ из программы на Java, и наоборот.

С помощью средств Java генерируется заголовочный файл (*.h), после чего создается файл с ассемблерной вставкой (*.cpp). Затем с помощью компилятора GCC из файла с ассемблерной вставкой собирается динамическая библиотека. Данная библиотека

подгружается с помощью статического метода System.load() из соответствующего класса.

```
public class TestJNI {
   public native static String bitwiseNot(String str);
   static {
      System.load("C:\\Users\\MyDLL.dll");
   }
   public TestJNI(){ }
}
```

Для вызова метода, подгружающего библиотеку с вычислением низкоуровневой функции, применяется механизм рефлексии. Рефлексия в Java осуществляется с помощью Java Reflection API. Рефлексия – это механизм исследования данных о программе во время её выполнения. Рефлексия позволяет исследовать информацию о полях, методах и конструкторах классов.

```
public void MyReflection(){
   strBitwise = textField1.getText();
   try {
      Class LowLevel = Class.forName("TestJNI");
      Constructor native_constructor = LowLevel.getConstructor();
      Method native_method = LowLevel.getMethod("bitwiseNot",String.class);
      Object natObj = native_constructor.newInstance();
      answer.setText((String) native_method.invoke(natObj,strBitwise));
   } catch (ClassNotFoundException | NoSuchMethodException | IllegalAccessException |
        InstantiationException | InvocationTargetException e) {
      e.printStackTrace();
      JOptionPane.showMessageDialog(LowView.this, "Выберите прилагаемую к СПО .dll",
          "Ошибка",JOptionPane.ERROR_MESSAGE);
      logForm.addAction("Загружена неверная .dll библиотека");
   }
}
```

В языке C# доступ к низкоуровневым вставкам реализуется через специализированный язык MSIL с использованием группы методов Emit. Для реализации модульности в отдельный проект вынесем реализацию низкоуровневой функции и создадим динамически подключаемую библиотеку .dll:

```
public class xorFunction
  {
    public static void xoring()
    {
      AppDomain asmDomain = Thread.GetDomain();
      AssemblyBuilder asmBuilder = asmDomain.DefineDynamicAssembly(new AssemblyName("XOR"), AssemblyBuilderAccess.RunAndSave);
      ModuleBuilder xorModule = asmBuilder.DefineDynamicModule("xoring.dll", true);
      TypeBuilder typeBuilder = xorModule.DefineType("XOR", TypeAttributes.Public);
      MethodBuilder methodBuilder = typeBuilder.DefineMethod("XOR", MethodAttributes.Public, typeof(int), new Type[] { typeof(int), typeof(int) });
      ILGenerator iLGenerator = methodBuilder.GetILGenerator();
      iLGenerator.Emit(OpCodes.Ldarg_1);
      iLGenerator.Emit(OpCodes.Ldarg_2);
      iLGenerator.Emit(OpCodes.Xor);
      iLGenerator.Emit(OpCodes.Ret);
      typeBuilder.CreateType();
      asmBuilder.Save("xoring.dll");
    }
  }
```

Для подключения созданной библиотеки применяется механизм рефлексии:

```
class FunctionLowController
  {
    private object obj;
    private MethodInfo m;
    private Logging log;
```

```
        public FunctionLowController(Logging _log)
        {
            log = _log;
            Assembly asm = Assembly.Load(File.ReadAllBytes("xoring.dll"));

    Type type = asm.GetType("XOR");
            m = type.GetMethod("XOR", BindingFlags.Instance | BindingFlags.Public);
            obj = Activator.CreateInstance(type);
        }

        public int getValueXOR(int val1, int val2)
        {
            log.addRecordToLog("Выполнена булева операция XOR");
            return (Int32)m.Invoke(obj, new object[] { val1, val2 });
        }
```

Лабораторная работа № 5
Вызов ассемблерных функций из языка высокого уровня

Цель работы – организация доступа к низкоуровневым операциям из программного обеспечения, написанного на языке высокого уровня.

Ход выполнения работы:

1) Изучить способы организации доступа к низкоуровневым операциям.

2) Написать код на низкоуровневом языке программирования.

3) Выделить код, написанный на низкоуровневом языке программирования, в отдельную динамически подключаемую библиотеку.

4) Вызвать на выполнение код, написанный на низкоуровневом языке программирования, из кода, написанного на языке программирования высокого уровня. При вызове кода использовать механизм рефлексии (Reflection).

Результаты работы:

Динамически подключаемая библиотека, содержащая код для доступа к низкоуровневым операциям.

3.9 Организация доступа к данным путем объектно-реляционного отображения

Совместное использование принципов объектно-ориентированного проектирования (ООП) с реляционными моделями данных (РМД) при разработке программной оболочки сложных системных программ ведет к необходимости использования средств объектно-реляционного отображения (рисунок 5). Основным достоинством такого подхода является манипуляция реляционными данными в терминах объектно-ориентированного подхода.

Рисунок 5 – Схема работы технологии ORM

Объектно-реляционное отображение (Object Relational Mapping, ORM) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Таблица (отношение, включая виртуальные таблицы) обычно соответствуют классу, строки таблицы – экземплярам этого класса, колонки таблицы при этом отображаются на атрибуты объекта или

вызовы методов чтения/записи. Отображение обычно двунаправленно: манипуляции с атрибутами объекта приводят к чтению информации из (и записи в) соответствующие таблицы базы данных. Могут также поддерживаться искусственные атрибуты (транслирующие значения в колонках таблиц), формульные атрибуты.

ORM обладают тремя основными достоинствами:

– явное описание схемы БД в терминах языка программирования существует и изменяется в одном месте;
– манипуляция привычными элементами языка программирования – классами, объектами, атрибутами и методами;
– автоматическая генерация SQL-запросов.

Разработаем для СПО организацию доступа к данным, предназначенным для функционирования, с использованием технологии-реляционного отображения. С помощью менеджера пакетов «NuGet» установим дополнительные библиотеки «NHiberanate» и «fluetnNHibernate». Модифицируем СПО, чтобы обеспечить использование объектно-реляционное отображение для доступа к данным. С помощью Framework «fluentNHibernate» реализуем доступ к хранилищу данных, используя CRUD(create, read, update, delete).

Создадим классы HibernateNotes и HibernateNotesJSON, которые будут хранить информацию об одной записи базы данных:

```
using System;

namespace Logical
{
   public class HibernateNotes
   {
      public virtual long Id { get; set; }
      public virtual string XMLServerAddress { get; set; }
      public virtual string XMLProtocol { get; set; }
```

```
public virtual int XMLPort { get; set; }
  }
}
```

Создадим маппинг-классы HibernateNotesMap и HibernateNotesJSONMap, которые необходимы для библиотеки NHibernate, чтобы выполнять операции с базой данных:

```
using FluentNHibernate.Mapping;

namespace Logical
{
  public class HibernateNotesMap : ClassMap<HibernateNotes>
  {
    public HibernateNotesMap()
    {
      Id(x => x.Id);
      Map(x => x.XMLServerAddress);
      Map(x => x.XMLProtocol);
      Map(x => x.XMLPort);
    }
  }
}
```

Для подключения к базе данных реализуем вспомогательный класс NHibernateHelper:

```
namespace Logical
{
  public class NHibernateHelper
  {
    public static ISession OpenSession()
    {
      ISessionFactory sessionFactory = Fluently.Configure()

.Database(FluentNHibernate.Cfg.Db.MsSqlConfiguration.MsSql2012
            .ConnectionString(String.Format(@"{0}", ConfigurationManag-
er.AppSettings.Get("WayDataBase"))).ShowSql())
          .Mappings(m                                                    =>
m.FluentMappings.AddFromAssemblyOf<HibernateNotes>())
```

```
            .ExposeConfiguration(cfg => new
SchemaUpdate(cfg).Execute(true, true))
            .BuildSessionFactory();

            return sessionFactory.OpenSession();
        }
    }
}
```

Строчку подключения к базе данных, необходимую для формирования конфигурационного файла вынесем в app.config.

Для работы с базой данных создадим интерфейсы IMHibernate и IMHibernateJSON, которые будут определять методы взаимодействия с БД:

```
using System.Collections.Generic;

namespace Logical
{
    public interface IMHibernate
    {
        void CreateResourceNote(HibernateNotes hibernateNotes);
        List<HibernateNotes> ReadResourceNote();
        void UpdateResourceNote(int index, HibernateNotes hibernateNotes);
        void DeleteResourceNote(int index);
        HibernateNotes GetAtElemnt(int index);
    }
}
```

Создадим класс MHibernate и MHibernateJSON, которые реализуют интерфейсы IModelHibernate и IMHibernateJSON. Данные классы реализуют систему CRUD (Create, read, update, delete):

```
using System.Collections.Generic;
using System.Linq;
using NHibernate;
using NHibernate.Transform;

namespace Logical
{
    public class MHibernate : IMHibernate
```

```
    {
        public MHibernate() { }

        public void CreateResourceNote(HibernateNotes hibernateNotes)
        {
            using (ISession s = NHibernateHelper.OpenSession())
            using (ITransaction tr = s.BeginTransaction())
            {
                s.Save(hibernateNotes);
                tr.Commit();
            }
        }
        public List<HibernateNotes> ReadResourceNote()
        {
            List<HibernateNotes> notes;

            using (var session = NHibernateHelper.OpenSession())
            {
                var queryBook = string.Format(@"select * from Hibernate-
Notes");
                notes = session.CreateSQLQuery(queryBook)

.SetResultTransformer(Transformers.AliasToBean(typeof(HibernateNotes)))
                .List<HibernateNotes>().ToList();
            }
            return notes;
        }

        public void UpdateResourceNote(int index, HibernateNotes hiber-
nateNotes)
        {
            using (ISession session = NHibernateHelper.OpenSession())
            using (ITransaction transaction = session.BeginTransaction())
            {
                session.Update(hibernateNotes);
                transaction.Commit();
            }
        }

        public void DeleteResourceNote(int index)
        {
```

```
            using (ISession s = NHibernateHelper.OpenSession())
            using (ITransaction tr = s.BeginTransaction())
            {
                s.Delete(s.Get<HibernateNotes>((long)index));
                tr.Commit();
            }
        }

        public HibernateNotes GetAtElemnt(int index)
        {
            HibernateNotes notes;
            using (ISession s = NHibernateHelper.OpenSession())
            {
                notes = s.Get<HibernateNotes>((long)index);
            }
            return notes;
        }
    }
}
```

Пример для Java

Для объектно-реляционного отображения используется фреймворк Hibernate ORM. Чтобы обозначить класс как сущность базы данных, его необходимо пометить специальными аннотациями:

```
@Entity
@Table(name = "records")
public class Record
{
    @Id
    @Column(name = "login", nullable = false)
    private String login;

    @Column(name = "password_hash", nullable = false)
    private String passwordHash;
    @Column(name = "email", unique = true, nullable = false)
    private String email;
}
```

Для создания и инициализации Hibernate сессии для работы с базой данных создается класс:

```
public final class HibernateUtil
{
   private static final SessionFactory sessionFactory;

   static{
      try{
         sessionFactory = new Configuration().configure().buildSessionFactory();
      }
      catch (Throwable ex) {
         System.err.println("Невозможно установить соединение" + ex);
         throw new ExceptionInInitializerError(ex);
      }
   }
```

При создании объекта типа RecordService происходит создание Hibernate-сессии, инициализация фреймворка, подключение к БД.

При вызове метода RecordService.loadFromDb() выполняется запрос к базе данных для извлечения информации о пользователях и изменение внутреннего состояния сервиса.

Лабораторная работа № 6
Организация доступа к данным путем объектно-реляционного отображения

Цель работы – организация доступа к данным, предназначенным для функционирования СПО, с использованием технологии объектно-реляционного отображения.

Ход выполнения работы:

1) Изучить принципы объектно-реляционного отображения.

2) Изучить один из фреймворков объектно-реляционного отображения (например, Hibernate, NHibernate, EF и т.п.)

3) Модифицировать СПО таким образом, чтобы обеспечить использование объектно-реляционного отображения для доступа к данным.

4) Модифицировать СПО таким образом, чтобы использовать специализированный язык запросов к данным, обеспечиваемый фреймворком объектно-реляционного отображения или технологической платформой (.NET LINQ, Java Streams и т.п.).

Результаты работы:

СПО, использующее технологию объектно-реляционного отображения для доступа к данным.

Примечание: СПО должно сохранять возможность работы с файлами.

3.10 Внедрение структурной обработки исключений

Тестирование, отладка и структурная обработка исключений при разработке СПО рассмотрены в разделе 2.6 настоящего учебного пособия.

Лабораторная работа № 7
Внедрение структурной обработки исключений

Цель работы – внедрение структурной обработки исключительных ситуаций в СПО.

Ход выполнения работы:

1) Изучить принципы структурной обработки исключений.

2) Разработать пользовательские исключения.

3) Модифицировать код СПО для внедрения структурной обработки исключений.

4) Внедрить систему логирования исключительных ситуаций.

Результаты работы:

СПО, обеспечивающее структурную обработку возникающих в процессе функционирования исключительных ситуаций.

Примечание: после внедрения структурной обработки исключений СПО не может выбрасывать необрабатываемые исключения. Все исключения должны обрабатываться.

3.11 Оценка эффективности функционирования системного программного обеспечения

Оценка эффективности СПО и алгоритмов его функционирования рассмотрена в разделе 2.5 настоящего учебного пособия.

Лабораторная работа № 8 Оценка эффективности функционирования системного программного обеспечения

Цель работы – оценка эффективности функционирования СПО и проведение оптимизации.

Ход выполнения работы:

1) Изучить принципы машинно-независимой оптимизации.
2) Изучить средства оценки эффективности функционирования – профайлеры (например, JetBrains dotMemory, dotTrace).
3) Изучить средства дизассемблирования программ (например, JetBrains dotPeek).
4) Выполнить дизассемблирование СПО и локализовать места в исходном коде СПО, которые подверглись машинно-независимой оптимизации.
5) Выполнить оценку эффективности функционирования СПО с использованием профайлеров памяти и процессорного времени.
6) В случае нахождения неэффективных программных кодов выполнить оптимизацию СПО по результатам проведенной оценки.
7) После выполнения оптимизации повторно выполнить оценку эффективности функционирования СПО с использованием профайлеров памяти и процессорного времени.

8) Сформировать документ, содержащий оценку эффективности функционирования СПО «до» и «после» оптимизации, а также описание выполненных модификаций программного кода.

Результаты работы:

Документ, описывающий процесс оценки и оптимизации СПО.

3.12 Документирование системного программного обеспечения

Виды программ и программных документов разрабатываются на основе ГОСТ 19.101-77 «Виды программ и программных документов». К программным документам относят документы, содержащие сведения, необходимые для разработки, изготовления, сопровождения и эксплуатации программ:

1) Спецификация ГОСТ 19.202-78 «Спецификация. Требования к содержанию и оформлению».
2) Ведомость держателей подлинников ГОСТ 19.403-79 «Ведомость держателей подлинников».
3) Текст программы ГОСТ 19.401-78 «Текст программы. Требования к содержанию и оформлению».
4) Описание программы ГОСТ 19.402-78 «Описание программы. Требования к содержанию и оформлению».
5) Программа и методика испытаний ГОСТ 19.301-79 «Программа и методика испытаний. Требования к содержанию и оформлению».
6) Техническое задание ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению».
7) Пояснительная записка ГОСТ 19.404-79 «Пояснительная записка. Требования к содержанию и оформлению».
8) Эксплуатационные документы.

К эксплуатационным документам относятся:

1) Ведомость эксплуатационных документов ГОСТ 19.507-79 «Ведомость эксплуатационных документов».

2) Формуляр ГОСТ 19.501-78 «Формуляр. Требования к содержанию и оформлению».
3) Описание применения ГОСТ 19.502-78 «Описание применения. Требования к содержанию и оформлению».
4) Руководство системного программиста ГОСТ 19.503-79 «Руководство системного программиста. Требования к содержанию и оформлению».
5) Руководство программиста ГОСТ 19.504-79 «Руководство программиста. Требования к содержанию и оформлению».
6) Руководство оператора ГОСТ 19.505-79 «Руководство оператора. Требования к содержанию и оформлению».
7) Описание языка ГОСТ 19.506-79 «Описание языка. Требования к содержанию и оформлению».
8) Руководство по техническому обслуживанию ГОСТ 19.508-79 «Руководство по техническому обслуживанию. Требования к содержанию и оформлению».

Лабораторная работа № 9
Документирование системного программного обеспечения

Цель работы – разработка программной документации для СПО.

Ход выполнения работы:

1) Изучить состав и структуру программной документации.
2) Подготовить «Руководство пользователя» для СПО.
3) Оформить отчет по циклу лабораторных работ.

Результаты работы:

Отчет по циклу лабораторных работ, описывающий результаты выполнения работ 1-8 и включающий в качестве приложения «Руководство пользователя».

Требования к результатам работы:

Отчет оформляется согласно СТО 02068410-004-2018 «Общие требования к учебным тестовым документам» [29].

Отчет на бумажном носителе печатается в одном экземпляре только с одной стороны листа. В местах, предназначенных для подписи студента, подписи должны быть проставлены до предоставления отчета преподавателю. Отчет предоставляется на проверку преподавателю в папке-скоросшивателе «под дырокол» без использования файлов/мультифор.

СПИСОК ЛИТЕРАТУРЫ

1 Аблязов, Р. Программирование на ассемблере на платформе x86-64 [Текст] / Р. Аблязов. – СПб.: ДМК Пресс, 2016. – 302 с.

2 Басс, Л. Архитектура программного обеспечения на практике [Текст] / Л. Басс, П. Клементс, Р. Кацман. – СПб.: Питер. – 2006. – 576 с.

3 Бхаргава, А. Грокаем алгоритмы [Текст] / А. Бхаргава – СПб.: Питер, 2018. – 288 с.

4 Вирт, Н. Алгоритмы и структуры данных [Текст] / Н. Вирт. – СПб.: ДМК Пресс, 2016. – 272 с.

5 Вирт, Н. Построение компиляторов [Текст] / Н. Вирт. – М.: ДМК Пресс, 2016. – 192 с.

6 Вирт, Н. Разработка операционной системы и компилятора. Проект Оберон [Текст] / Н. Вирт, Ю. Гуткнехт. – М.: ДМК Пресс, 2012. – 560 с.

7 Головнин, О.К. Программирование на языке высокого уровня с элементами системного анализа информации : учебное пособие / О.К. Головнин, А.А. Столбова. – Самара : Изд-во Самарского университета, 2019. – 132 с.

8 ГОСТ 2.105-95 Единая система конструкторской документации. Общие требования к текстовым документам [Текст]. – Введ. 1996-06-30. – М: Издательство стандартов, 1995. – 31 с.

9 Иванова, Н.Ю. Системное и прикладное программное обеспечение : учебное пособие [Текст] / Н.Ю. Иванова, В.Г. Маняхина. – Москва: Прометей, 2011. – 202 с.

10 Истомин, Е.П. Высокоуровневые методы информатики и программирования : учебник для вузов [Текст] / Е.П. Истомин. – СПб.: Андреевский издательский дом, 2008. – 228 с.

11 Канцедал, С.А. Алгоритмизация и программирование : учеб. пособие для ссузов [Текст] / С.А. Канцедал. – М.: Инфра-М, 2008. – 351 с.

12 Кнут, Д.Э. Искусство программирования. Том 1. Основные алгоритмы / Д.Э. Кнут. – М.: Вильямс, 2017. – 720 с.

13 Компиляторы. Принципы, технологии и инструментарий [Текст] / А.В. Ахо, М.С. Лам, Р. Сети, Дж.Д. Ульман. – М.: Диалектика-Вильямс, 2017. – 1184 с.

14 Курячий, Г.В. Операционная система Linux. Курс лекций. Учебное пособие [Текст] / Г.В. Курячий, К.А. Маслинский. – М.: ДМК Пресс, 2010. – 348 с.

15 Макконелл, Дж. Анализ алгоритмов. Активный обучающий подход [Текст] / Дж. Макконелл. – М.: Техносфера, 2013. – 415 с.

16 Макконнелл, С. Совершенный код [Текст] / С. Макконнелл. – СПб.: БХВ, 2017. –896 с.

17 Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения [Текст] / Р. Мартин. – СПб.: Питер, 2018. – 352 с.

18 Николаев, Е.И. Объектно-ориентированное программирование : учебное пособие [Текст] / Е.И. Николаев. – Ставрополь: СКФУ, 2015. – 225 с.

19 Окулов, С.М. Программирование в алгоритмах [Текст] / С.М. Окулов. – М.: Бином, 2013. – 384 с.

20 Олифер, В.Г. Сетевые операционные системы [Текст] / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер, 2002. – 544 с.

21 Павловская, Т.А. C#. Программирование на языке высокого уровня [Текст] / Т.А. Павловская. – СПб.: Питер, 2014. – 432 с.

22 Портал разработчиков Microsoft Developer Network [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru.

23 Приемы объектно-ориентированного проектирования. Паттерны проектирования [Текст] / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб.: Питер, 2008. – 366 с.

24 Рефакторинг. Улучшение проекта существующего кода [Текст] / М. Фаулер, К. Бек, Дж. Брант, Д. Робертс. – СПб.: Вильямс, 2017. – 448 с.

25 Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# [Текст] / Дж. Рихтер. – СПб.: Питер, 2019. – 896 с.

26 Солдатов, В.П. Программирование драйверов Windows [Текст] / В.П. Солдатов. – М.: Бином, 2014. – 576 с.

27 Сообщество разработчиков программного обеспечения [Электронный ресурс]. – Режим доступа: https://stackoverflow.com.

28 Сорокина, С.И. Программирование драйверов и систем безопасности : учебное пособие [Текст] / С.И. Сорокина, А.Ю. Тихонов, А.Ю. Щербаков. – СПб.: БХВ-Петербург, 2003. – 256 с.

29 СТО 02068410-004-2018 Общие требования к учебным текстовым документам [Текст]. – Самара: Самарский университет, 2018. – 36 с.

30 Столбова, А.А. Теоретические основы и практические аспекты информатики и программирования для решения задач управления и обработки информации на языке C# : учебное пособие / А.А. Столбова, О.К. Головнин. – Самара : Изд-во Самарского университета, 2019. – 164 с.

31 Структуры данных и алгоритмы [Текст] / А.В. Ахо, Дж.Э. Хопкрофт, Дж.Д. Ульман. – М.: Вильямс, 2018. – 400 с.

32 Таненбаум, Э. Современные операционные системы [Текст] / Э. Таненбаум. – СПб.: Питер, 2005. – 1038 с.

33 Теория и реализация языков программирования : учеб. пособие по курсу теории и реализации яз. Программирования [Текст] / В.А. Серябриков [и др.]. – М.: МЗ Пресс, 2006. – 348 с.

34 Фаулер, М. Архитектура корпоративных программных приложений [Текст] / М. Фаулер. – М.: Вильямс, 2006. – 544 с.

35 Чеповский, А. Common Intermediate Language и системное программирование в Microsoft .NET [Текст] / А. Чеповский, А. Макаров, С. Скоробогатов. – М.: ИНТУИТ, 2016. – 399 с.

ПРИЛОЖЕНИЕ А

Список теоретических вопросов к экзамену

1) Системные программы: основные понятия и определения, классификация и структура.

2) Системное программное обеспечение: операционные системы, загрузчики, драйверы, системы программирования, трансляторы, компиляторы и интерпретаторы, отладчики, утилиты.

3) Драйвера устройств.

4) Трансляторы. Общая схема работы.

5) Ассемблеры, компиляторы и интерпретаторы. Назначение и принципы построения.

6) Компиляция. Фазы компиляции: лексический, синтаксический и семантический анализ, оптимизация, генерация кода, сборка.

7) Системы программирования. Структура системы программирования. Функционирование системы программирования.

8) Инструментальные средства разработки системных программ.

9) Процесс проектирования системного программного обеспечения.

10) Библиотеки подпрограмм. Динамически подключаемые библиотеки DLL. Диспетчеры пакетов.

11) Машинно-независимая оптимизация программ. Оптимизация кода. Рефакторинг.

12) Оценка эффективности функционирования системного программного обеспечения. Профайлинг.

13) Архитектура системных программ. Многоуровневая архитектура. Организация межпрограммных связей.

14) Метапрограммирование.

15) Многопоточная работа.

16) Работа с памятью.

17) Работа с файлами.

18) Лямбда-исчисление и язык интегрированных запросов.

19) Вызов ассемблерных функций из языка высокого уровня.

20) Разработка системного программного обеспечения с использованием принципов SOLID и паттернов проектирования.

21) Обработка исключений. Перехват исключительных ситуаций и их корректная обработка. Политики обработки исключений.

22) Документирование системного программного обеспечения.

ПРИЛОЖЕНИЕ Б

Задание на контрольную работу

Задание

Контрольная работа носит реферативный характер и заключается в написании реферата на индивидуальную тему в текстовом процессоре (Microsoft Word, OpenOffice Writer, LibreOffice Writer).

Выбор варианта работы

Индивидуальную тему работы студент выбирает по номеру варианта, который определяется двумя последними цифрами номера зачетной книжки студента. Если две последние цифры номера зачетной книжки превышают число вариантов работы, то номер варианта определяется путем сложения двух последних цифр номера зачетной книжки студента.

Примеры:

13 (такой вариант есть) = 13 вариант;

26 (такого варианта нет, складываем) = 2 + 6 = 8 вариант;

99 (такого варианта нет, складываем) = 9 + 9 = 18 (такого варианта нет, значит снова складываем) = 1 + 8 = 9 вариант.

Защита работы

Контрольная работа защищается во время занятий или консультаций. Защита проходит в форме устного опроса, форма отчетности – «зачет/незачет». Контрольная работа может быть возвращена на доработку при наличии существенных замечаний по содержанию материала, несоответствии темы работы ее содержанию, несоответствии индивидуального варианта подготовленной работе, несоответствии оформления работы требованиям СТО Самарского университета 02068410-004-2018 «Общие требования к учебным текстовым документам».

Форма представления работы

Работа предоставляется на электронную почту преподавателя в формате текстового процессора (Microsoft Word, OpenOffice Writer, LibreOffice Writer), используемого для подготовки работы, и на бумажном носителе в одном экземпляре. Работа на бумажном носителе печатается в одном экземпляре только с одной стороны листа. Реферат предоставляется на проверку преподавателю в папке-скоросшивателе «под дырокол» без использования файлов/мультифор.

Требования к оформлению работы

Контрольная работа оформляется согласно СТО Самарского университета 02068410-004-2018 «Общие требования к учебным текстовым документам».

Содержание работы

Контрольная работа должна содержать следующие структурные элементы в указанном порядке:

– титульный лист (1 страница);
– содержание;
– введение (1-2 страницы);
– основная часть (2-3 главы);
– заключение (1-2 страницы);
– список использованных источников (не менее 5 источников).

При необходимости, работа может быть дополнена разделом «Приложения». Минимальное количество листов в работе – 15, максимальное – 30. Листы из раздела «Приложения» не учитываются. Желательно наличие иллюстраций, примеров.

Варианты заданий для контрольной работы

1) Архитектурные модели системного программного обеспечения.

2) Взаимодействие разработчиков. Системы контроля версий.

3) Диспетчеры пакетов.

4) Объектно-реляционное отображение (ORM).

5) Паттерны проектирования системного программного обеспечения.

6) Переменные окружения.

7) Производительность системного программного обеспечения. Эффективность алгоритмов.

8) Промежуточное программное обеспечение (middleware).

9) Разработка на платформе .NET. Общеязыковая исполняющая среда CLR. Язык программирования CIL (MSIL). Общая система типов Common Type System .NET.

10) Сервисные программы (утилиты).

11) Системы отслеживания ошибок.

12) Системы программирования: назначение, состав, классификация.

13) Службы Microsoft Windows: назначение, режимы работы, обзор популярных служб.

14) Тестирование, отладка, обработка исключительных ситуаций в формализме системного программного обеспечения.

15) Управление проектами и задачами. Программные средства управления проектами и задачами.

16) Шаблоны запросов и обновления данных. Язык интегрированных запросов.

ПРИЛОЖЕНИЕ В

Пример оформления отчета

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра информационных систем и технологий

ОТЧЁТ
по циклу лабораторных работ
по курсу «Системное программирование»
Вариант 11

Выполнил: Васечкин В.В.,
гр. 6301-090301D

Проверил: к.т.н., доцент кафедры информационных систем и технологий Столбова А.А.

Самара 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева» (Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра информационных систем и технологий

«УТВЕРЖДАЮ»
Руководитель лабораторных работ
__________________Столбова А.А.
____ ___________________ 2021 г.

ЗАДАНИЕ
на цикл лабораторных работ по курсу
«Системное программирование»
студенту группы 6301-090301D
Васечкин Василий Васильевич

1) Исходные данные к лабораторной работе определяются согласно индивидуальному варианту: 11.
2) Перечень вопросов, подлежащих разработке в ходе выполнения лабораторных работ, приведен в Приложении к заданию.
3) Срок завершения цикла лабораторных работ: 31 мая 2021 г.
4) Дата выдачи задания: 17 февраля 2021 г.
5) График выполнения цикла лабораторных работ:

Лабораторная работа	Срок выполнения	Итоги проверки	
		Отметка о выполнении	Подпись преподавателя и дата
1. Анализ требований к СПО	01.03.2021		
2. Проектирование СПО	01.03.2021		

3. Создание сложной структуры данных	16.03.2021		
4. Разработка СПО с использованием принципов SOLID и паттернов проектирования	19.03.2021		
5. Вызов ассемблерных функций из языка высокого уровня	26.03.2021		
6. Организация доступа к данным путем объектно-реляционного отображения	30.03.2021		
7. Внедрение структурной обработки исключений	02.04.2021		
8. Оценка эффективности функционирования СПО	09.04.2021		
9. Документирование СПО	13.04.2021		

Задание принял к исполнению:
____ _____________ ___ 2021 г. _________________ / В. В. Васечкин

ПРИЛОЖЕНИЕ
к заданию

Цель цикла лабораторных работ – получение практических навыков проектирования, разработки, отладки, тестирования и документирования системного программного обеспечения (СПО).

1 Требования к информационному обеспечению

Руководство пользователя должно быть оформлено в соответствии с ГОСТ 19.505-79 «Руководство оператора. Требования к содержанию и оформлению».

Отчет о выполнении цикла лабораторных работ должен быть составлен в соответствии с СТО 02068410-004-2018 «Общие требования к учебным текстовым документам».

2 Функции, реализуемые разрабатываемым СПО

2.1 СПО должно работать с файлами формата XML.

Файл должен содержать следующие данные: адрес сервера, порт, протокол.

СПО должно иметь возможность сохранения файла и загрузки из файла введенных пользователем данных.

СПО должно обеспечивать возможность добавления, удаления, редактирования записей в файле.

При добавлении или редактировании записи должен быть предусмотрен ручной ввод адреса сервера и порта, протокол должен выбираться из заданного списка: TCP, UDP.

Данные из файла должны выводиться на экран в табличном виде.

2.2 СПО должно обеспечить разбор и анализ следующих вводимых с клавиатуры конструкций языка:

– итеративный цикл:

for(<инициализация>;<условие>;<итератор>){<тело цикла>}.

СПО должно обеспечивать лексический, синтаксический и семантический анализ конструкции языка.

Анализ конструкции должен быть реализован на основе автоматной грамматики.

Должна быть предоставлена возможность называть управляющую переменную любым допустимым именем.

В качестве управляющей переменной должна задаваться переменная типа int.

Должна быть возможность добавить в тело цикла любые операторы, поддерживаемые языком программирования.

Ошибки, обнаруженные при анализе конструкции языка должны выводиться на экран.

СПО должно вывести на экран ответ на вопрос: сколько раз выполнится цикл.

2.3 СПО должно обеспечить выполнение низкоуровнего кода с передачей параметров для следующих функций:

– сложение двух целых чисел со знаком с проверкой переполнения.

Результат сложения должен выводиться на экран.

Реализация функции должна быть выполнена на низкоуровневом языке.

3 Требования к программному обеспечению

Операционная система: Windows 7 и выше.

4 Требования к аппаратному обеспечению

Процессор: Intel(R) Core(TM).

Установленная память (ОЗУ): 1 ГБ.

Тип системы: 64-разрядная операционная система.

5 Требования к языкам программирования и средам разработки СПО

Языки программирования: C#, MSIL.

Среда разработки: Visual Studio 2017.

6 Требования к структуре СПО:

СПО в минимальной реализации должно состоять из следующих элементов:

– исполняемый файл с расширением .exe, содержащий элементы графического пользовательского интерфейса, программный код для организации человеко-машинного взаимодействия и программный код, обращающийся к динамически подключаемым библиотекам;

– динамически подключаемая библиотека с расширением .dll, содержащая вызов низкоуровневых функций, которая должна вызываться с использованием механизма рефлексии;

– динамически подключаемая библиотека с расширением .dll, содержащая бизнес-логику приложения.

7 Требования к способам организации диалога с пользователем:

СПО должно иметь графический пользовательский интерфейс, не нарушающий принятые подходы к проектированию интерфейсов.

8 Требования к качеству реализации функций:

СПО должно обеспечивать корректную работу с вводимыми данными.

СПО должно обеспечивать обработку исключительных ситуаций в случае их возникновения.

СПО должно формировать записи текущей работы (логи) и выводить их на экран.

9 Требования к быстродействию: отсутствуют.

Руководитель лабораторных работ:

____________________ / А. А. Столбова

Задание принял к исполнению: _____________ / В. В. Васечкин

СОДЕРЖАНИЕ

Лабораторная работа № 1 Проектирование системного программного обеспечения 115

Лабораторная работа №2 Создание сложной структуры данных 119

Лабораторная работа № 3 Разработка системного программного обеспечения с использованием принципов SOLID и паттернов проектирования 127

Лабораторная работа № 4 Вызов ассемблерных функций из языка высокого уровня 138

Лабораторная работа № 5 Организация доступа к данным путем объектно-реляционного отображения 142

Лабораторная работа № 6 Внедрение структурной обработки исключений 146

Лабораторная работа № 7 Оценка эффективности функционирования системного программного обеспечения 148

Заключение 157

Список использованных источников 158

Приложение А Руководство пользователя 159

Лабораторная работа № 1
Проектирование системного программного обеспечения

Системное программное обеспечение включает в себя следующие модули (рисунок 1):

1 MainWindow;
2 DllAnalyzer;
3 DllAsm;
4 ChangeWindow.

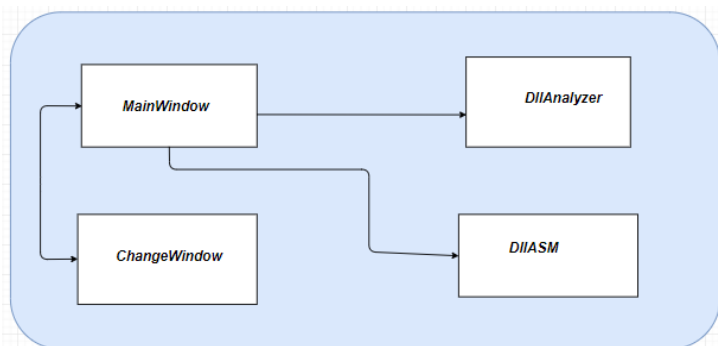

Рисунок 1 – Структурная схема СПО

На рисунке 2 представлена диаграмма классов системного программного обеспечения.

Модуль MainWindow состоит из:

1 OpenFile – функция, отвечающая за открытие файла. Она состоит из поля path и метода OpenFileDialog. Path – поле, отвечающее за хранение пути файла. OpenFileDialog – метод, отвечающий за открытие диалогового окна открытия файла.

2 SaveFile – функция, отвечающая за сохранение файла. Она состоит из поля path и метода SaveFileDialog. Path – поле, отвечающее за хранение пути файла. SaveFileDialog – метод, отвечающий за открытие диалогового окна сохранения файла.

3 DeleteFile – функция, отвечающая за удаление файла. Она состоит из поля path и метода Delete. Path – поле, отвечающее за хранение пути файла. Delete – метод, отвечающий за удаление файла.

4 InfoFile – функция, отвечающая за получение информации о файле. Она состоит из поля path и методов GetCode, GetPath, GetSize и GetDate. Path – поле, отвечающее за хранение пути файла. GetCode – метод, отвечающий за получение бинарного кода файла. GetPath – метод, отвечающий за получение полного пути файла. GetSize– метод, отвечающий за получение размера файла. GetDate – метод, отвечающий за получение даты создания файла.

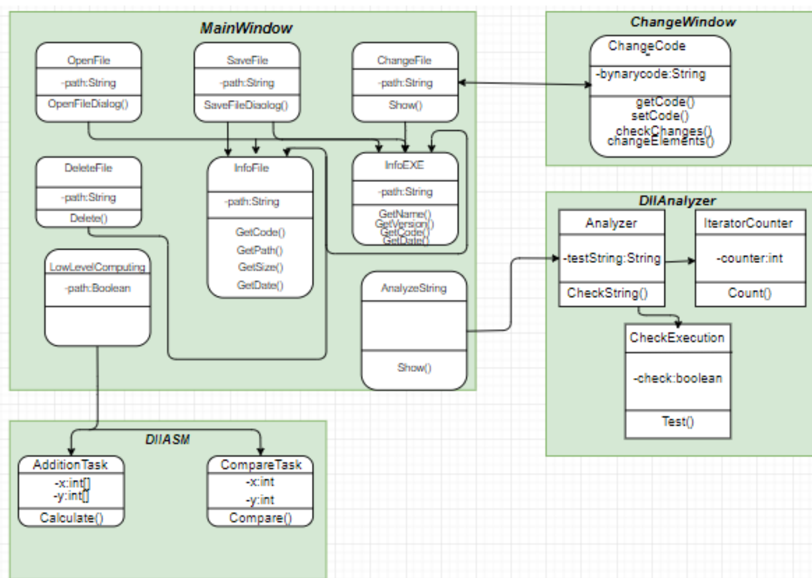

Рисунок 2 – Диаграмма классов СПО

5 LowLevelComputing– функция, отвечающая за выбор низкоуровневой функции. Она состоит из поля selector и метода Show. Selector – поле, отвечающее за выбор низкоуровневой функции.

Show – метод, отвечающий за открытие окна выполнения низкоуровневой функции.

6 AnalyzeString – функция, отвечающая за открытие окна анализатора пользовательской строки. Она состоит из метода Show. Show – метод, отвечающий за открытие окна анализатора пользовательской строки.

Библиотека DllAnalyzer включает в себя:

1 Analyzer – функция, отвечающая анализ пользовательской строки. Она состоит из поля teststring и метода CheckString. TestString – поле, отвечающее за хранение пользовательской строки. CheckString – метод, отвечающий за анализ пользовательской строки.

2 IterationCounter – функция, отвечающая за подсчёт количества выполненных циклов. Она состоит из поля counter и метода Count. Counter – поле, отвечающее за хранение количества выполненных циклов. Count – метод, отвечающий за подсчёт количества выполненных циклов.

3 CheckExecution – функция, отвечающая за проверку выполнения цикла больше одного раза. Она состоит из поля check и метода Test. Check – поле, отвечающее за факт выполнения цикла. Test – метод, определяющий факт выполнения цикла.

Библиотека DllAsm состоит из:

1 AdditionTask – функция, отвечающая за разделение одного значения без знака на другое значение без знака. Она состоит из полей x и y, отвечающих за хранение в массиве каких-то значений и метода Calculate, отвечающего за разделение значений без знака.

2 CompareTask – функция, отвечающая за сравнение двух значений с условием «строго больше». Она состоит из полей x и y, отвечающих за хранение двух значений и метода Compare, отвечающего за сравнения этих значений.

Модуль ChangeWindow включает в себя:

1 ChangeCode – функция, отвечающая за изменение файла JSON. Она состоит из поля JSONFile и методов GetCode, SetCode, CheckChanges, ChangeElement. GetCode – метод, отвечающий за получение бинарного кода. SetCode – метод, отвечающий за установку изменённого бинарного кода. CheckChanges – метод, определяющий факт изменения файла. ChangeElement – метод, отвечающий за изменение отдельного элемента бинарного кода.

Системные требования, необходимые для работы СПО:

- Microsoft Windows 10;
- Microsoft .NET Framework 4.5.1.

Требования к аппаратному обеспечению:

1) Процессор: AMD A10-8700P Radeon R6 10 Compute cores 4C+6G 1.80 GHz.

2) ОЗУ: 8.0 Гб.

Лабораторная работа № 2
Создание сложной структуры данных

Для реализации работы с файлами типа text\plain были разработаны следующие классы: MyFile и Note.

Класс Note используется для реализации структуры данных, описывающей одну запись о настройках сервера. В нем содержатся следующие поля:

– address – предназначено для хранения записи об адресе сервера;

– port – предназначено для хранения номера порта;

– protocol – протокол передачи данных между клиентом и сервером в сети

Класс MyFile предназначен для реализации методов работы над объектами класса RecordPlainText. Класс содержит поле path, в котором хранится путь к файлу, и следующие методы:

– Update – отвечает за изменение записи в файле по выбранному номеру записи;

– Output_to_dgv – отвечает за вывод данных из файла в таблицу;

– AddToEnd – отвечает за добавление в файл новой записи;

– DelForIndex – отвечает за удаление записи из файла;

Продемонстрируем выбор и открытие файла для последующей работы с ним. Для этого необходимо нажать кнопку «Выбрать файл», после чего появится окно для выбора файла, показанное на рисунке 3.

Продемонстрируем попытку вывода данных в таблицу в том случае, если файл не был выбран. Для этого не будем осуществлять выбор файла, показанный на рисунке 3, и сразу нажмем на кнопку «Вывести данные».

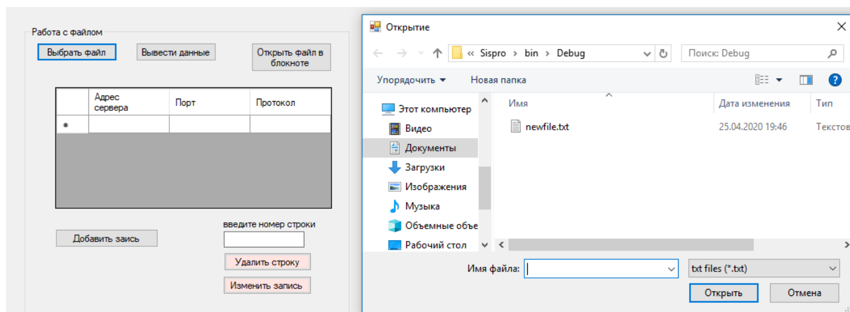

Рисунок 3 – Демонстрация работы кнопки «Выбрать файл»

После попытки вывести данные появится сообщение об ошибке, представленное на рисунке 4.

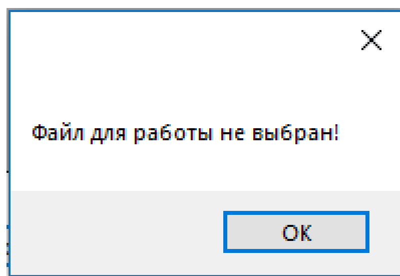

Рисунок 4 – Сообщение об ошибке, если не выбран файл

Для демонстрации выполнения вывода данных откроем файл newfile.txt и нажмем на кнопку «Вывести данные». Результат выполнения вывода данных показан на рисунке 5.

Продемонстрируем попытку открытия файла в блокноте с помощью notepad.exe в том случае, если файл не был выбран. Для этого не будем осуществлять выбор файла и сразу нажмем на кнопку «Открыть файл в блокноте». После попытки открытия файла, появится сообщение об ошибке. Для демонстрации выполнения открытия файла в блокноте откроем файл newfile.txt и нажмем на кнопку «Открыть файл в блокноте».

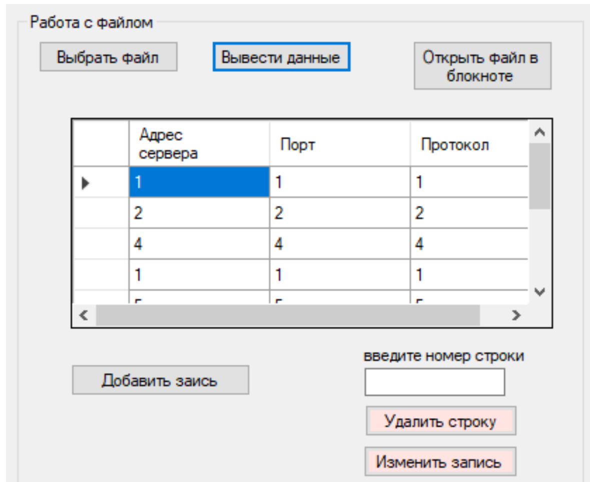

Рисунок 5 – Демонстрация работы кнопки «Вывести данные»

Результат выполнения открытия файла в блокноте показан на рисунке 6.

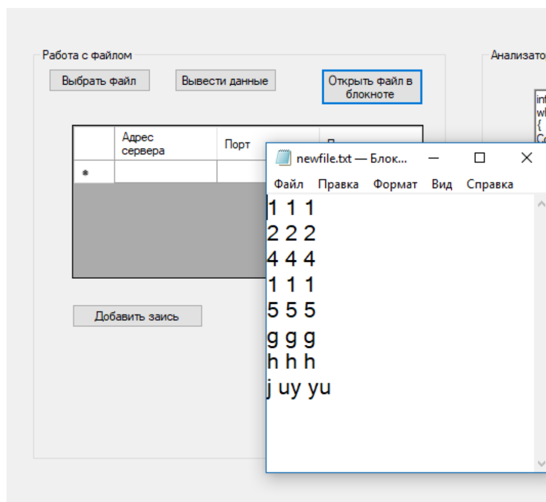

Рисунок 6 – Демонстрация работы кнопки «Открыть файл в блокноте»

Продемонстрируем попытку добавления новой записи в файл в том случае, если файл не был выбран. Для этого не будем осуществлять выбор файла и сразу нажмем на кнопку «Добавить запись». После попытки добавления записи появится сообщение об ошибке. Если выбор файла был успешно осуществлен, при нажатии на кнопку «Добавить запись» появится окно для ввода данных новой записи, представленное на рисунке 7.

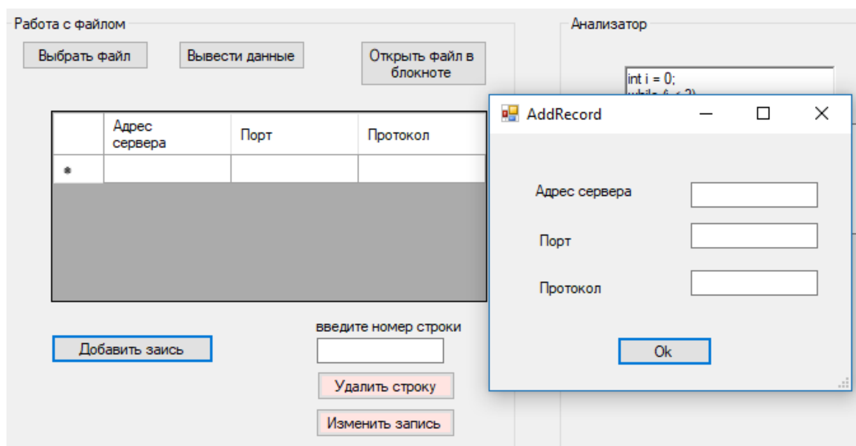

Рисунок 7– Окно ввода данных

Для демонстрации выполнения добавления записи в файл введем следующие значения в поля для ввода (рисунок 8):

– адрес сервера – NewData;
– порт – NewData2;
– протокол – NewData3.

Результат выполнения добавления новой записи представлен на рисунке 9.

Продемонстрируем выполнение удаления строки из файла по номеру. Для этого не будем осуществлять выбор файла и сразу нажмем на кнопку «Удалить строку». После попытки удаления

строки появится сообщение об ошибке. Если выбор файла был осуществлен, но в поле для номера строки был введен некорректный номер строки, или же номер строки не был введен вовсе, появится сообщение об ошибке, которое показано на рисунке 10.

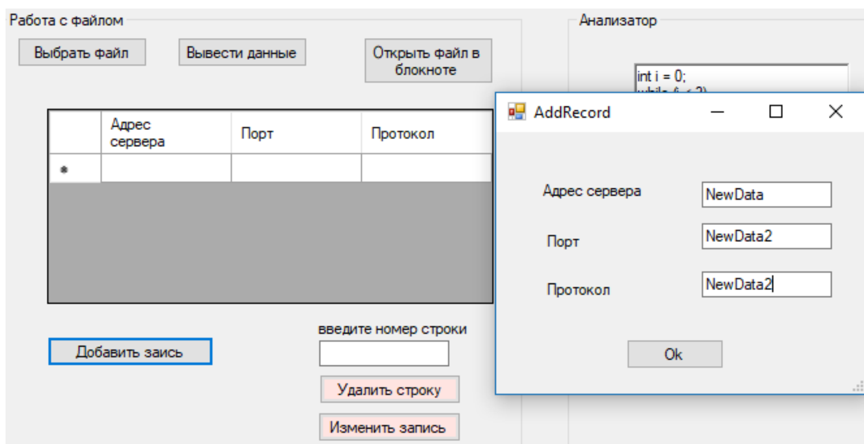

Рисунок 8 – Ввод данных

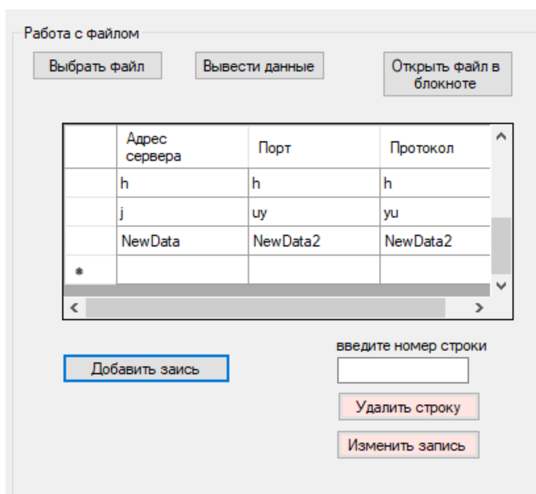

Рисунок 9 – Результат добавления записи

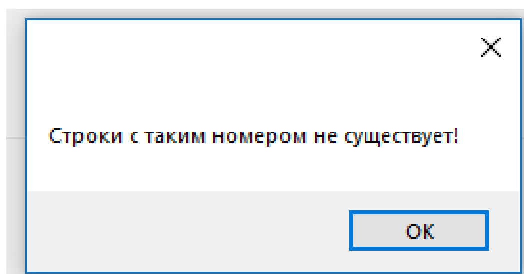

Рисунок 10 – Ошибка при попытке удалить строку с неверным номером

Для демонстрации успешного выполнения удаления строки по номеру введем значение «1» в поле для ввода, как показано на рисунке 11, а затем нажмем на кнопку «Удалить строку».

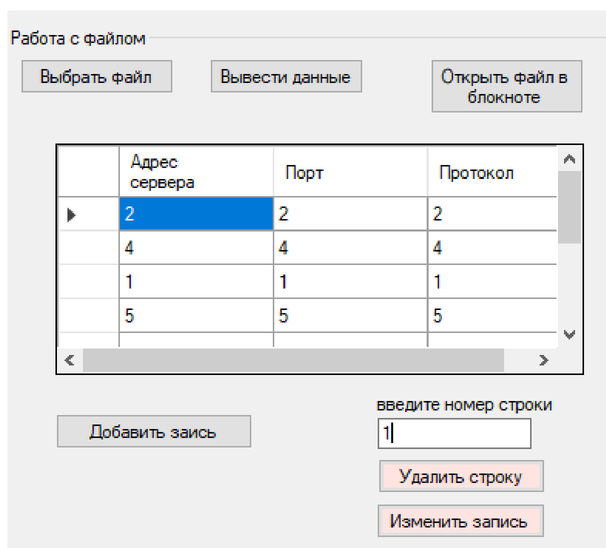

Рисунок 11 – Ввод номера удаляемой строки

Результат удаления строки продемонстрирован на рисунке 12.

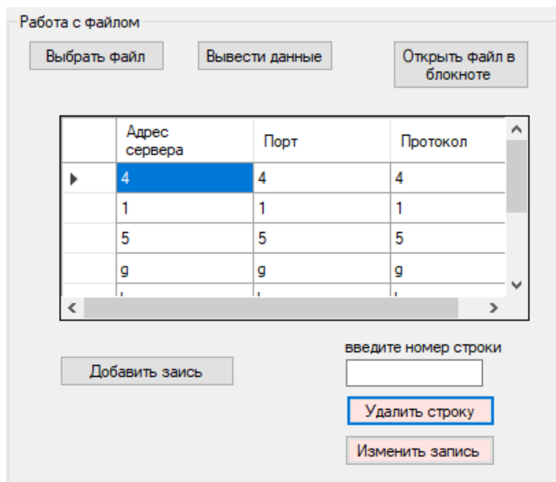

Рисунок 12 – Результат удаления строки из файла

Продемонстрируем выполнение изменения строки из файла по номеру. Для этого не будем осуществлять выбор файла и сразу нажмем на кнопку «Изменить запись». После попытки удаления строки появится сообщение об ошибке. Если выбор файла был осуществлен, но в поле для номера строки был введен некорректный номер строки, или же номер строки не был введен вовсе, появится сообщение об ошибке. Для демонстрации успешного выполнения изменения строки по номеру введем значение «2» в поле для ввода, а затем нажмем на кнопку «Изменить запись». После нажатия на кнопку «Изменить запись» появится окно для ввода новых значений полей (рисунок 13).

Для демонстрации выполнения добавления записи в файл введем следующие значения в поля для ввода:

– адрес сервера – 111;
– порт – 1111;
– протокол – 11111.

Результат изменения строки продемонстрирован на рисунке 14.

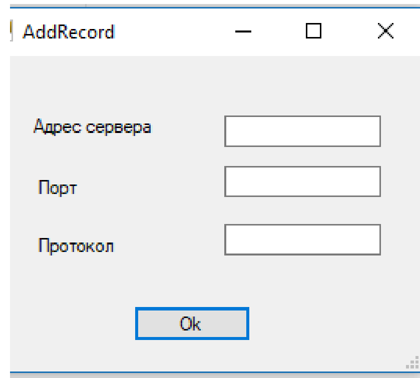

Рисунок 13 – Окно для ввода измененной строки

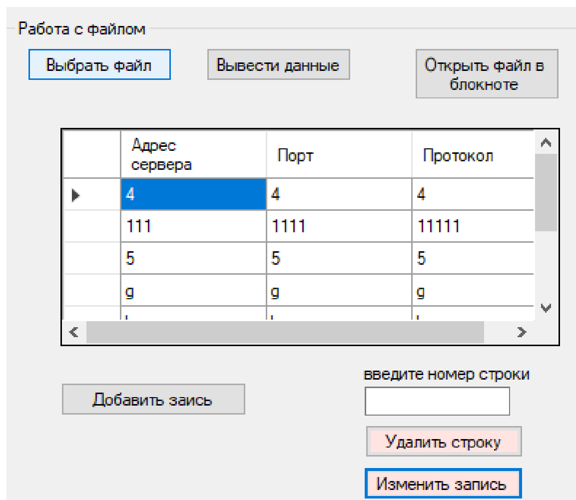

Рисунок 14 – Результат изменеия строки

Лабораторная работа № 3
Разработка системного программного обеспечения с использованием принципов SOLID и паттернов проектирования

В процессе проектирования СПО были изучены принципы SOLID. SOLID – это акроним, образованный из заглавных букв пяти принципов ООП и проектирования:

1) SRP – принцип единственной ответственности, который гласит, что должно быть не более одной причины изменить класс.

2) Принцип OCP – принцип открытости и закрытости, предполагает возможности расширения сущностей, но не изменения.

3) Третий принцип LSP завязан на механизме наследования, когда объекты можно заменить их наследниками.

4) Следующий принцип ISP. Его смысл – разделение «толстых» интерфейсов, но более мелкие.

5) Последний принцип DIP, когда модули верхних уровней не зависят от модулей нижних уровней.

Для разработки программного кода применен паттерн проектирования MVP, использующий при программировании принцип разделения кода на части:

– Модель (англ. Model) — данные для отображения;

– Вид (англ. View) — реализует отображение данных (из Модели), обращается к Presenter за обновлениями, перенаправляет события от пользователя в Presenter;

– Представитель (англ. Presenter) — реализует взаимодействие между Моделью и Видом и содержит в себе всю бизнес-логику; при необходимости получает данные из хранилища и преобразует для отображения во View.

Для реализации паттерна был создан интерфейс IMainView, в котором определено поведение модели (листинг 1).

Листинг 1 – Фрагмент интерфейса IMainView

```
event EventHandler AddNodeButtonEvent;
event EventHandler DeleteNodeButtonEvent;
event EventHandler SaveToFileButtonEvent;
event EventHandler ReadFromFileButtonEvent;
event EventHandler ChangeNodeButtonEvent;
event EventHandler CalcButtonEvent;
event EventHandler ClearButtonEvent;
event EventHandler AnalyseButtonEvent;
event EventHandler AddButtonBDEvent;
event EventHandler DeleteButtonBDEvent;
event EventHandler SetButtonBDEvent;
event EventHandler SaveButtonBDEvent;
event EventHandler OpenButtonBDEvent;
sList list { get; set; }
string analyseString { get; }
string result { get; set; }
string fileName { get; }
string fileText { get; }
void show();
void myShowDialog();
DataGridView data { get; set; }
```

Для реализации основной логики программы был создан интерфейс IModel (листинг 2).

Листинг 2 – Интерфейс IModel

```
namespace BisLogic
{
   public interface IModel
   {
      Analyz analyz { get; set; }
      void Insert(string login, string passHash, string email);
```

```
        void Delete(int pos);
        void Save(string fileName);
        void Read(string fileName);
        void PrintGrid(DataGridView data);
        void Change(int pos, int what, string newInfo);
        int Size();
        void newList();
        string start(string a);
        string r { get; set; }
        string result { get; set; }
    }
}
```

Класс Model реализует интерфейс IModel и содержит реализацию функций, определяющих поведение программы (листинг 3).

Листинг 3 – Класс Model

```
public class ModMain : IModel
    {
        private sList list;
        public Analyz analyz { get; set; }
        public string r { get; set; }
        public string result { get; set; }
        public ModMain(sList list, Analyz analyz)
        {
            this.list = list;
            this.analyz = analyz;
            this.r = analyz.R;
            this.result = analyz.Result;
        }
        public void Delete(int pos) { list.Delete(pos); }
        public void Insert(string login, string passHash, string email) {
list.Insert(login, passHash, email); }
```

```
public void Read(string fileName) { sFileWorking.Read(fileName, list); }
public void Save(string fileName) { sFileWorking.Save(fileName, list); }
public void PrintGrid(DataGridView data) { list.PrintGrid(data); }
public void Change(int pos, int what, string newInfo) { list.Change(pos, what, newInfo); }
public int Size() { return list.Size(); }
public void newList(){list=new sList();}
public string start(string a){ result = analyz.start(a); r = analyz.R; return result; }

}
```

Для определения взаимодействия между моделью и видом был создан класс PMainPresenter (листинг 4).

Листинг 4 – фрагмент класса PMainPresenter

```
private readonly IMainView view;
private readonly IModel model;
private readonly AddPresenter addPresenter;
public MainPresenter(IMainView view, IModel model, AddPresenter addPresenter)
{
    this.view = view;
    this.model = model;
    this.addPresenter = addPresenter
    view.AnalyseButtonEvent += new EventHandler(AnalyzeButton);
    view.AddNodeButtonEvent += new EventHandler(AddButton);
    view.DeleteNodeButtonEvent += new EventHandler(DeleteButton);
```

```
        view.ChangeNodeButtonEvent += new EventHandler(ChangeButton);
        view.SaveToFileButtonEvent += new EventHandler(SaveButton);
        view.ReadFromFileButtonEvent += new EventHandler(OpenButton);
        view.CalcButtonEvent += new EventHandler(CalcButton);
        view.ClearButtonEvent += new EventHandler(ClearButton);
        LogsMessage.tb = view.LogTextBox;
        LogsMessage.tb2 = view.LogTextBox2;
        LogsMessage.tb3 = view.LogTextBox3;
    }
```

Для построения модели необходимо использование интерфейсов, так как непосредственное использование классов при развитии приложения может повлечь некоторые трудности.

Реализуем интерфейс IFile:

```
interface IFile
{
    string PathText { get; }
    DataGridView GetDgv { get; }
    string GetStr { get; }
    TextBox GetTextbox { get; }
    String AddLog { set; }
}
```

Реализуем интерфейс ICompiler:

```
interface ICompiler
{
    TextBox GetTextboxC { get; }
    RichTextBox GetRichTextbox { get; }
    Button GetB8 { get; }
    Label Getlabel { get; }
```

```
    String AddLog { set; }
}
```

Реализуем интерфейс ILowFunction:

```
interface ILowFunction
{
    ComboBox GetCombo1 { get; }
    ComboBox GetCombo2 { get; }
    string SetText {  set; }
    String AddLog { set; }}
```

Создадим форму MainForm, которая будет представлять основную графическую составляющую программы (рисунок 15).

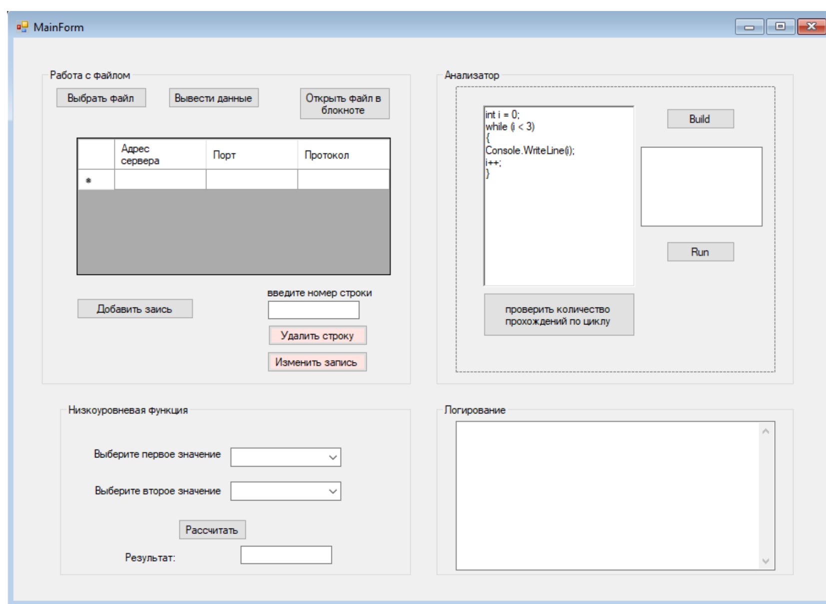

Рисунок 15 – Графический интерфейс программы

В форме MainForm представлены следующие элементы:

– textBox – для отображения и ввода текстовой информации;

– richtextBox – для отображения и ввода текстовой информации;

– comboBox – для отображения и ввода текстовой информации;

– groupBox – для предоставления идентифицируемого группирования для других элементов управления;

– label – для отображения информации о полях ввода и разграничения интерфейса;

– button – для запуска запрограммированных действий в программе;

– dataGridView – для вывода в табличном представлении данных из файла.

Создадим вспомогательную форму AddRecord, реализующую добавление записей в таблицу dataGridView (рисунок 16).

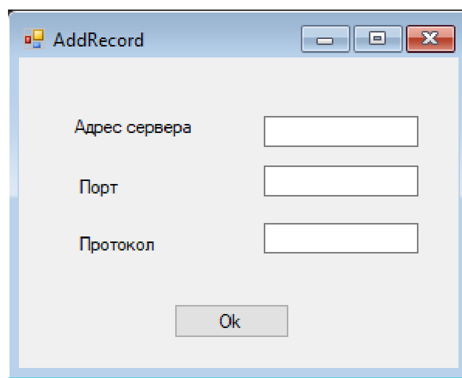

Рисунок 16 – Форма для добавления новой записи настроек сервера

В форме AddRecord представлены следующие элементы:

– textBox – для отображения и ввода текстовой информации;

– label – для отображения информации о полях ввода и разграничения интерфейса;

– button – для запуска запрограммированных действий в программе;

Продемонстрируем программную обработку синтаксически неверного кода. Введем следующий код в поле для ввода:

```
int i = 0;
whiled (i < 3)
{
   Console.WriteLine(i);
   i++;
}
```

После попытки скомпилировать код в поле для отображения ошибок выведется сообщение об обнаруженной ошибке (рисунок 17).

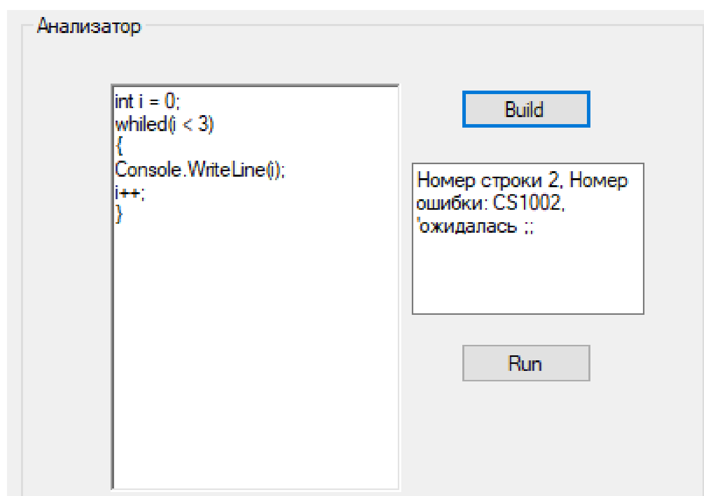

Рисунок 17 – Демонстрация компиляции программы с синтаксически неверным кодом

Продемонстрируем программную обработку синтаксически верного кода, не содержащего конструкцию while(<условие>) {<те-

ло цикла>} соответствующую индивидуальному заданию. Введем следующий код в поле для ввода:

```
for (int i=0; i<4; i++)
{
   Console.WriteLine(i);
   i++;
}
```

После попытки скомпилировать код появится сообщение об ошибке.

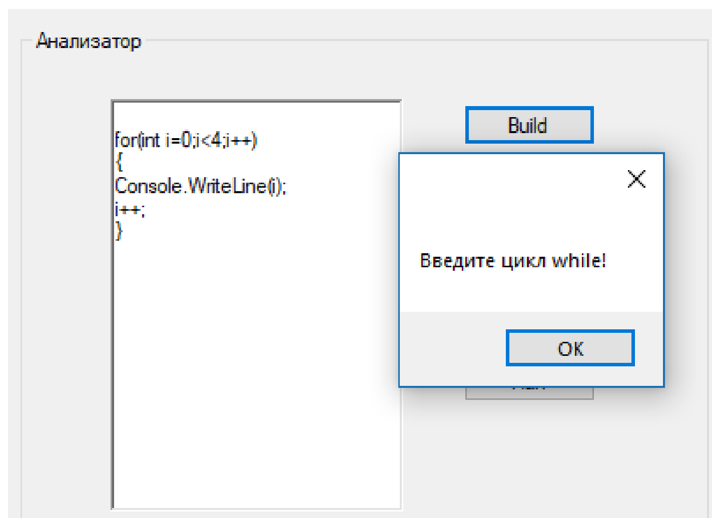

Рисунок 18 – Демонстрация компиляции программы с верным кодом, не содержащем конструкцию while (<условие>) {<тело цикла>}, соответствующую индивидуальному заданию

Для демонстрации успешной компиляции кода введем следующий код в поле для ввода:

```
int i = 0;
while (i < 3)
```

```
{
   Console.WriteLine(i);
   i++;
}
```

Результат выполнения продемонстрирован на рисунке 19.

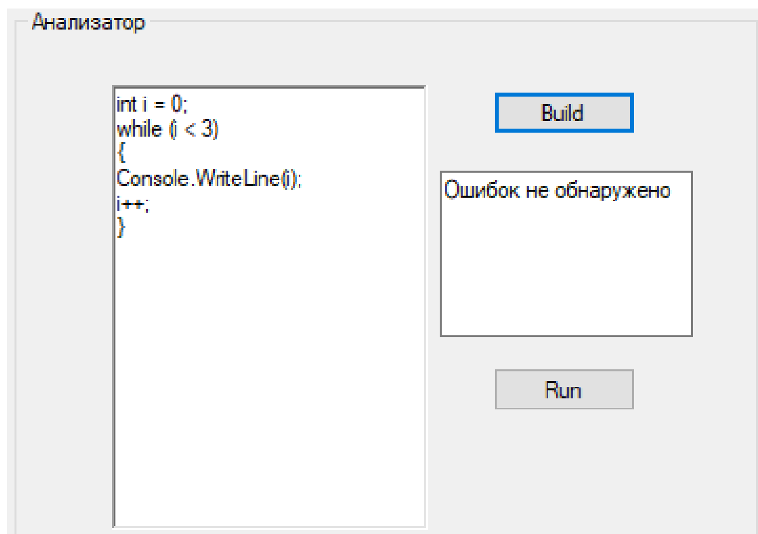

Рисунок 19 – Демонстрация компиляции программы с синтаксически верным кодом

Для демонстрации подсчета количества прохождений по циклу необходимо нажать на кнопку «Проверить количество прохождений по циклу», после чего появится информация о количестве прохождений по циклу, продемонстрированная на рисунке 20.

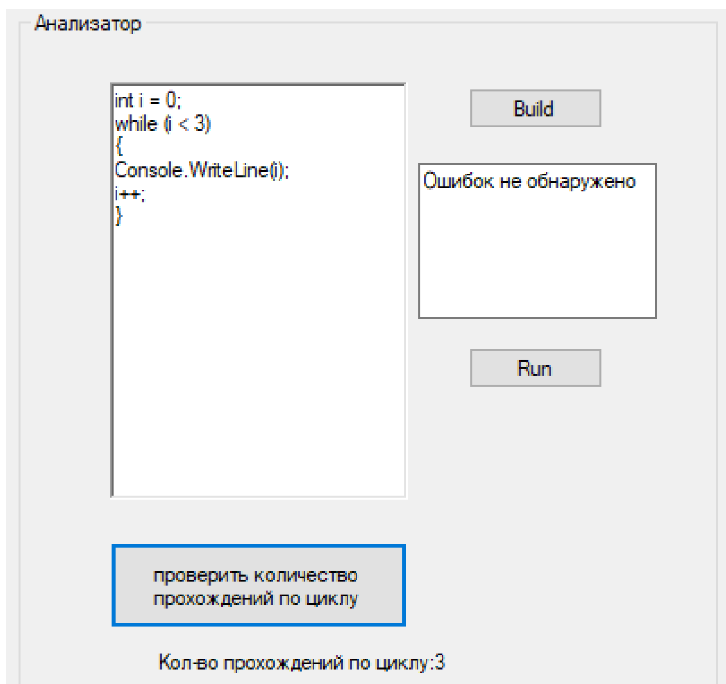

Рисунок 20 – Подсчет количества прохождений

Лабораторная работа № 4
Вызов ассемблерных функций из языка высокого уровня

Организуем доступ к низкоуровневым операциям из программного обеспечения, написанного на языке высокого уровня. Выделим код, написанный на низкоуровневом языке программирования, в отдельную динамически подключаемую библиотеку AsmFunc.dll. Библиотека AsmFunc.dll состоит из одного класса: Func.cs.

Класс Func.cs:

```
using System;
using System.Runtime.InteropServices;

namespace AsmFunc
{
   public class Func
   {
      [DllImport("kernel32.dll")]
      static extern bool VirtualProtect(IntPtr lpAddress, uint size, uint
flNewProtect, out uint lpflOldProtect);
      delegate int asmdeleg(int a, int b);
      public int GetFunc(int a, int b)
      {
         byte[] codeasm = { 0x8B, 0x44, 0x24, 0x04, 0x8B, 0x54, 0x24,
0x08, 0x09, 0xD0, 0xC2, 0x08, 0x00 };
         IntPtr adresasm = Marshal.AllocHGlobal(codeasm.Length);
         uint oldprotect;
         VirtualProtect(adresasm, (uint)codeasm.Length, 0x40, out
oldprotect);
         Marshal.Copy(codeasm, 0, adresasm, codeasm.Length);
         asmdeleg injectcall = (asmdel-
eg)Marshal.GetDelegateForFunctionPointer(adresasm,
typeof(asmdeleg));
```

```
            return injectcall(a,b);
        }
    }
}
```

Заполним форму Form3.cs в СПО, написанного на языке высокого уровня, и используем механизм рефлексии при вызове низкоуровневой функции.

Класс Form3.cs

```
using System;
using System.Reflection;
using System.Windows.Forms;

namespace SP_Ganeev_11
{
    public partial class Form3 : Form
    {
        public Form3()
        { InitializeComponent(); }

        private void button1_Click(object sender, EventArgs e)
        {
            Assembly asm = Assembly.LoadFrom("AsmFunc.dll");
            Type myType = asm.GetType("AsmFunc.Func",true);
            object obj = Activator.CreateInstance(myType);
            MethodInfo method = myType.GetMethod("GetFunc");
            object d = method.Invoke(obj, new object[] {
Int32.Parse(textBox1.Text), Int32.Parse(textBox2.Text) });
            textBox3.Text = d.ToString();
        }
    }
}
```

Вызовем на выполнение низкоуровневую функцию, из кода, написанного на языке программирования высокого уровня. После запуска программы отобразится главное окно программы в соответствии с рисунком 21:

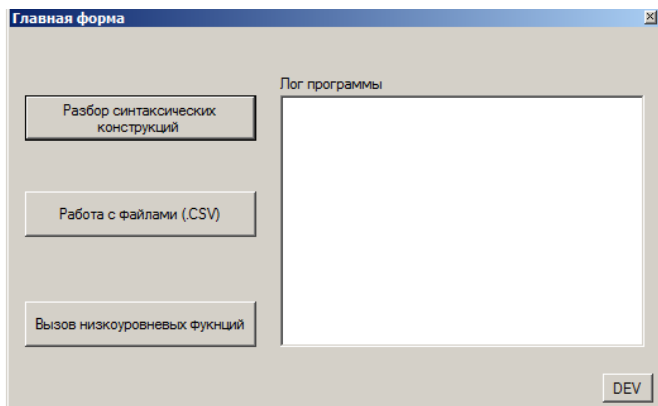

Рисунок 21 – MajorForm.cs

При нажатии на кнопку «Вызов низкоуровневых функций» откроется окно, представленное на рисунке 22.

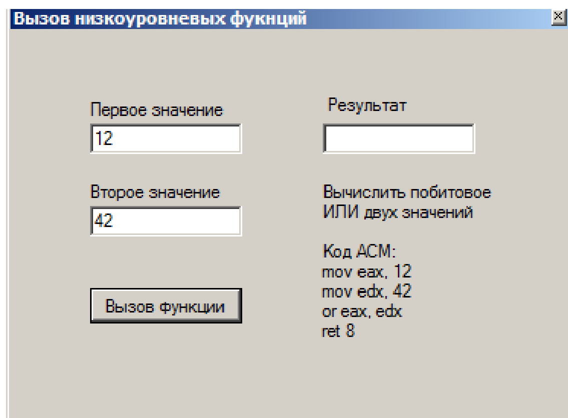

Рисунок 22 – Fom3.cs

Введём первое и второе значения. Нажмём кнопку «Вызов функции». Результат представлен на рисунке 23.

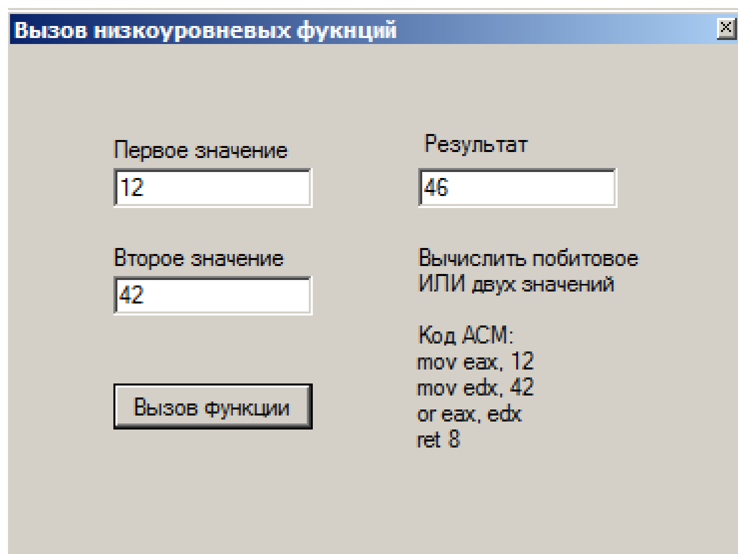

Рисунок 23 – Действие программы после нажатия кнопки «Вызов функции»

Лабораторная работа № 5
Организация доступа к данным путем объектно-реляционного отображения

Для разработки организации доступа СПО к данным, предназначенным для функционирования СПО, с использование технологии объектно-реляционного отображения был использован фреймворк «Entity Framework», подключенный с помощью менеджера пакетов «NuGet».

Для подключения к базе данных в файл конфигурации App.config внесена вся необходимая информация о СУБД.

Внесенные изменения в файл App.config:

```
<connectionStrings>
<add name="DbConnectionString" connectionString="Data Source=DESKTOP-UKG970G\SQLEXPRESS;
    Initial Catalog=DBName2;Integrated Security=True;
    " providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Кроме того, были созданы специальные классы для работы с базами данных – класс DbContext и класс Table, описывающий структуру нашей таблицы:

```
public class MyDBContext: DbContext
{
    public MyDBContext() : base("DbConnectionString")
    { }
    public DbSet<Table> Tables { get; set; }
}
public class Table
{
    public int Id { get; set; }
```

```
    public string Addres_Servera { get; set; }
    public string Port { get; set; }
    public string Protocol { get; set; }
}
```

Добавлен новый метод в класс для работы с файлом MyFile:

```
    public void AddToDB(string str)
    {
        string[] mystring = str.Split(' ');
        using (var table = new MyDBContext())
        {

            var sql = new Table()
            {
                Addres_Servera = mystring[0],
                Port = mystring[1],
                Protocol = mystring[2]
            };
            table.Tables.Add(sql);
            table.SaveChanges();
            MessageBox.Show(sql.Addres_Servera);                Message-
Box.Show(sql.Port); MessageBox.Show(sql.Protocol);
        }
        }
```

Продемонстрируем работу с базой данных. Для этого нажмем на кнопку «Добавить запись», заполним поля для ввода новыми значениями, нажмем на кнопку «Ok». Добавление данных продемонстрировано на рисунке 24.

Результат добавления данных в файл и таблицу отображен на рисунке 25.

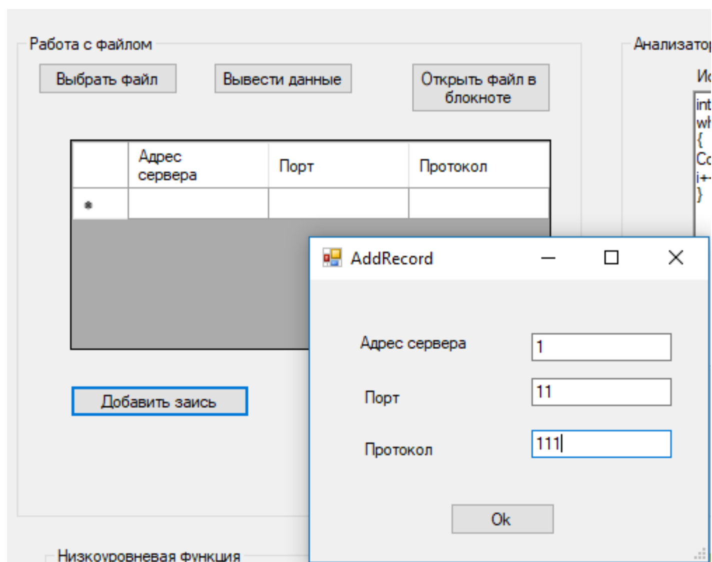

Рисунок 24 – Демонстрация добавления данных

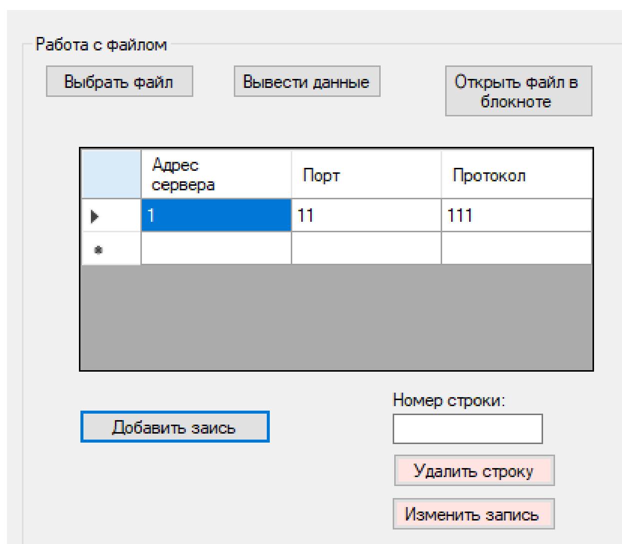

Рисунок 25 – Данные, добавленные в DataGridView

Далее перейдем в SQL Server Management Studio и удостоверимся, что база данных и таблица были созданы (рисунок 26).

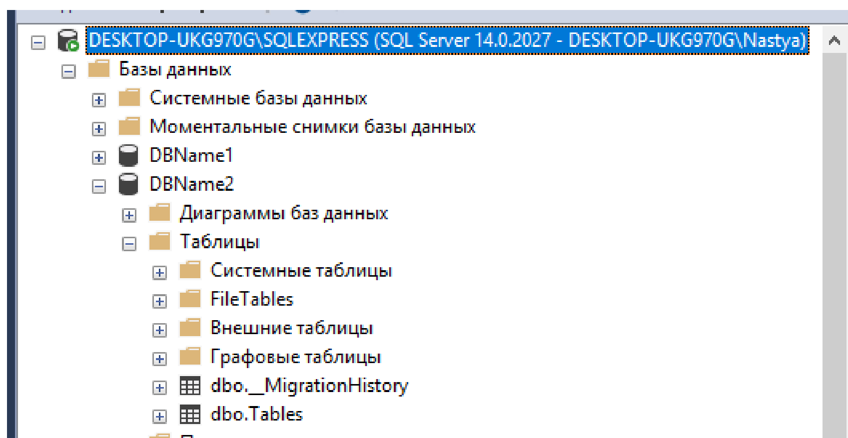

Рисунок 26 – Созданные БД и таблица Tables

Выполним запрос «select *from Tables» для отображения содержимого созданной таблицы, представленного на рисунке 27.

Результаты | Сообщения

	Id	Addres_Servera	Port	Protocol
1	1	1	11	111

Рисунок 27 – Содержимое таблицы Tables

Лабораторная работа № 6
Внедрение структурной обработки исключений

Разработаем пользовательское исключение. Класс FileException реализующий Exception:

```
class FileException : Exception
{
    public FileException(string message) : base(message)
    {
    }
}
```

Модифицируем методы классов и добавим выбросы исключения при вводе некорректных значения. Обработка исключения:

```
/// <summary>
        /// Метод для записи в файл
        /// </summary>
        /// <param name="records"></param>
        /// <param name="file"></param>
        /// <returns></returns>
        public static bool write(RecordList records, string file)
        {
            try
            {
                var write = new ChoCSVWriter<Record>(file);
                List<Record> rec = records.getList();
                write.Write(rec);
                write.Flush();
                return true;
            }
            catch(FileNotFoundException)
            {
                    return false;
```

```
            throw new FileException("Файл не найден");
        }
}
```

Результат работы программы представлен на рисунке 28.

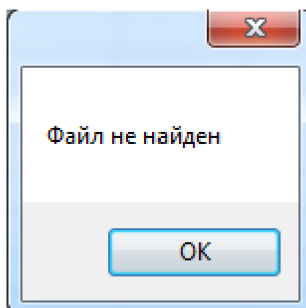

Рисунок 28 – Результат обработки исключений

Лабораторная работа № 7
Оценка эффективности функционирования системного программного обеспечения

Оценим эффективность функционирования СПО и проведём оптимизацию. Выберем сторонний профайлер, а именно EQATEC Profiler. Выполним оценку эффективности функционирования СПО до оптимизации и получим результаты в четвёртой колонке «Old Avg(Full)».

Проведём оптимизацию СПО, модифицировав программный код:

1) В Form2.CsvWriter выражения получения количества столбцов. ColumnCount вынесем из циклов в локальную переменную.

Код ДО:

```
public void CsvWriter(string way)
{
   try
   {
      StreamWriter sw = new StreamWriter(way, false, Encoding.Default);
      for (int i = 0; i < dataGridView1.ColumnCount; i++)
      {
         sw.Write(dataGridView1.Columns[i].Name);
         if (i < dataGridView1.ColumnCount - 1)
            sw.Write(',');
      }
      sw.WriteLine();
      for (int i = 0; i < dataGridView1.RowCount; i++)
      {
         for (int j = 0; j < dataGridView1.ColumnCount; j++)
         {
            sw.Write(dataGridView1[j, i].Value);
```

```
                if (j < dataGridView1.ColumnCount - 1)
                    sw.Write(',');
            }
            sw.WriteLine();
        }
        sw.Close();
    }
    catch (Exception e)
    {
        MessageBox.Show("Ошибка в блоке записи CSV файла");
        LogException.WriteLog(e, "Ошибка в блоке записи CSV 
файла");
    }
}
```

Код ПОСЛЕ:

```
public void CsvWriter(string way)
    {
        try
        {
            StreamWriter sw = new StreamWriter(way, false, Encod-
ing.Default);
            int cc = dataGridView1.ColumnCount;
            for (int i = 0; i < cc; i++)
            {
                sw.Write(dataGridView1.Columns[i].Name);
                if (i < cc - 1)
                    sw.Write(',');
            }
            sw.WriteLine();
            for (int i = 0; i < dataGridView1.RowCount; i++)
```

```
            {
               for (int j = 0; j < cc; j++)
               {
                  sw.Write(dataGridView1[j, i].Value);
                  if (j < cc - 1)
                     sw.Write(',');
               }
               sw.WriteLine();
            }
            sw.Close();
         }
         catch (Exception e)
         {
            MessageBox.Show("Ошибка в блоке записи CSV файла");
            LogException.WriteLog(e, "Ошибка в блоке записи CSV файла");
         }
 }
```

2) В Form2.inputB_Click удалим повторное создание экземпляра класса FileInfo.

Код ДО:

```
 private void inputB_Click(object sender, EventArgs e)
  {
     try
     {
        MajorForm major = this.Owner as MajorForm;
        major.listAdd(inputB.Text);
        if (dataGridView1.Enabled == false)
        {
           MessageBox.Show("Загрузите .CSV файл");
```

```
            }
            else
            {
               if (openFileDialog2.ShowDialog() == DialogResult.OK)
               {
                  FileInfo fileEx = new FileInfo(openFileDialog2.FileName);
                  string fe = fileEx.Extension;
                  if (fe == ".exe")
                  {
                     string a, b, c = null;
                     FileInfo fileIn = new FileInfo(openFileDialog2.FileName);
                     FileVersionInfo fileversion = FileVersionInfo.GetVersionInfo(openFileDialog2.FileName);
                     a = fileIn.Name;
                     b = fileversion.FileVersion;
                     c = fileIn.LastWriteTime.ToShortDateString();
                     list.Add(a + ',' + b + ',' + c);
                     UpdateTable();
                     UpdateList();
                  }
                  else throw new MyException("Выберите файл с расширением ", " .EXE");
               }
            }
         }
         catch (MyException ex)
         {
            MessageBox.Show(ex.Message + "-" + ex.FileEx);
            LogException.WriteLog(ex, ex.Message + "-" + ex.FileEx);
```

```
        }
        catch (Exception ex)
        {
            MessageBox.Show("Ошибка в блоке загрзуки EXE файла (диалоговое окно)");
            LogException.WriteLog(ex, "Ошибка в блоке загрзуки EXE файла (диалоговое окно)");
        }
    }
```

Код ПОСЛЕ:

```
private void inputB_Click(object sender, EventArgs e)
    {
        try
        {
            MajorForm major = this.Owner as MajorForm;
            major.listAdd(inputB.Text);
            if (dataGridView1.Enabled == false)
            {
                MessageBox.Show("Загрузите .CSV файл");
            }
            else
            {
                if (openFileDialog2.ShowDialog() == DialogResult.OK)
                {
                    FileInfo fileEx = new FileInfo(openFileDialog2.FileName);
                    string fe = fileEx.Extension;
                    if (fe == ".exe")
                    {
                        string a, b, c = null;
```

```
                    FileVersionInfo fileversion = FileVersionIn-
fo.GetVersionInfo(openFileDialog2.FileName);
                    a = fileEx.Name;
                    b = fileversion.FileVersion;
                    c = fileEx.LastWriteTime.ToShortDateString();
                    list.Add(a + ',' + b + ',' + c);
                    UpdateTable();
                    UpdateList();
                }
                else throw new MyException("Выберите файл с расши-
рением ", " .EXE");
            }
        }
    }
    catch (MyException ex)
    {
        MessageBox.Show(ex.Message + "-" + ex.FileEx);
        LogException.WriteLog(ex, ex.Message + "-" + ex.FileEx);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Ошибка в блоке загрзуки EXE файла
(диалоговое окно)");
        LogException.WriteLog(ex, "Ошибка в блоке загрзуки EXE
файла (диалоговое окно)");
    }
}
```

3) В Form2.deleteB_Click удалим лишние логические вычисления. Вместо двух if сделаем один.

Код ДО:

```
private void deleteB_Click(object sender, EventArgs e)
    {
        try
        {
            MajorForm major = this.Owner as MajorForm;
            major.listAdd(deleteB.Text);
            if (dataGridView1.Enabled == false)
            {
                MessageBox.Show("Загрузите .CSV файл");
            }
            else
            {
                if (dataGridView1.CurrentRow == null)
                {
                    MessageBox.Show("Выделите строку для удаления");
                }
                else
                {
                    list.DeleteIndex(dataGridView1.CurrentRow.Index);
                    UpdateTable();
                    UpdateList();
                }
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Ошибка в блоке удаления записи");
            LogException.WriteLog(ex, "Ошибка в блоке удаления за-
писи");
        }
}
```

Код ПОСЛЕ:

```
private void deleteB_Click(object sender, EventArgs e)
    {
        try
        {
            MajorForm major = this.Owner as MajorForm;
            major.listAdd(deleteB.Text);

            if (dataGridView1.CurrentRow == null)
            {
                MessageBox.Show("Загрузите .CSV файл или выдели-
те строку для удаления");
            }
            else
            {
                list.DeleteIndex(dataGridView1.CurrentRow.Index);
                UpdateTable();
                UpdateList();
            }

        }
        catch (Exception ex)
        {
            MessageBox.Show("Ошибка в блоке удаления записи");
            LogException.WriteLog(ex, "Ошибка в блоке удаления за-
писи");
        }
}
```

Выполним повторную оценку эффективности функционирования СПО и получим результаты в пятой колонке “New Avg(Full)”.

Из результатов видно, что общее количество времени выполнения стало меньше на 1644 мс, то есть оптимизация СПО прошла успешно (рисунок 29).

Confidence	Speedup	Diff Avg(full)	Old Avg(full)	New Avg(full)	Diff Calls	Old Calls	New Calls	Method name
	1.06 x	-1 644	28 445	26 801	0	1	1	SP_Ganeev_11.Program.Main
	1.08 x	-176	2 487	2 311	0	1	1	SP_Ganeev_11.Form2.saveB_Click(System.Object,System.EventArgs)
	1.09 x	-149	1 895	1 746	0	1	1	SP_Ganeev_11.Form2.inputB_Click(System.Object,System.EventArgs)
	1.04 x	-79	2 205	2 126	0	1	1	SP_Ganeev_11.Form2.downloadB_Click(System.Object,System.EventArgs)
	1.57 x	-4.0	11	7.0	0	1	1	SP_Ganeev_11.Form1.Dispose(System.Boolean)
	1.50 x	-2.0	6.0	4.0	0	1	1	SP_Ganeev_11.MajorForm.Dispose(System.Boolean)
	1.25 x	-1.0	5.0	4.0	0	1	1	SP_Ganeev_11.Form1.ctor
	2.00 x	-1.0	2.0	1.0	0	1	1	SP_Ganeev_11.Form2.deleteB_Click(System.Object,System.EventArgs)
	1.04 x	-1.0	24	23	0	1	1	SP_Ganeev_11.Form2.CsvWriter(System.String)
	1.00 x	0	1.0	1.0	0	1	1	SP_Ganeev_11.Form2.UpdateTable
	1.00 x	0	20	20	0	1	1	SP_Ganeev_11.Form2.Dispose(System.Boolean)
	1.00 x	0	2.0	2.0	0	1	1	SP_Ganeev_11.MajorForm.ctor
	1.00 x	0	1.0	1.0	0	2	2	SP_Ganeev_11.Form2.CsvReader(System.String)
	1.00 x	0	6.0	6.0	0	1	1	SP_Ganeev_11.Form3.ctor
	1.00 x	0	12	12	0	1	1	SP_Ganeev_11.Form3.button1_Click(System.Object,System.EventArgs)
	1.00 x	0	2.0	2.0	0	1	1	SP_Ganeev_11.Form2.editB_Click(System.Object,System.EventArgs)
	1.00 x	0	21	21	0	1	1	SP_Ganeev_11.MajorForm.button1_Click(System.Object,System.EventArgs)
	1.00 x	0	17	17	0	1	1	SP_Ganeev_11.MajorForm.button3_Click(System.Object,System.EventArgs)
	1.00 x	0	14	14	0	1	1	SP_Ganeev_11.MajorForm.InitializeComponent
	0.83 x	+0.3	1.3	1.5	0	4	4	SP_Ganeev_11.Form2.UpdateList
	0.70 x	+0.3	0.6	0.9	0	11	11	SP_Ganeev_11.MajorForm.listAdd(System.String)
	0.00 x	+1.0	0	1.0	0	1	1	SP_Ganeev_11.Form1.InitializeComponent
	0.83 x	+1.0	5.0	6.0	0	1	1	SP_Ganeev_11.Form3.InitializeComponent
	0.81 x	+3.0	13	16	0	1	1	SP_Ganeev_11.Form2.ctor
	0.93 x	+4.0	53	57	0	1	1	SP_Ganeev_11.Form1.button2_Click(System.Object,System.EventArgs)
	0.56 x	+4.0	5.0	9.0	0	1	1	SP_Ganeev_11.Form2.InitializeComponent
	0.64 x	+4.0	7.0	11	0	1	1	SP_Ganeev_11.Form3.Dispose(System.Boolean)
	0.76 x	+10	31	41	0	1	1	SP_Ganeev_11.MajorForm.button2_Click(System.Object,System.EventArgs)
	0.33 x	+155	77	232	0	1	1	SP_Ganeev_11.Form1.button1_Click(System.Object,System.EventArgs)

Рисунок 29 – Оценка эффективности функционирования СПО

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторных работ были изучены порядок проектирования СПО, методология проектирования структур данных, некоторые принципы и паттерны программирования, способы организации доступа к низкоуровневым операциям, принципы объектно-реляционного отображения, фреймворк объектно-реляционного отображения NHibernate, принципы структурной обработки исключений, машинно-независимой оптимизации и структура программной документации, а также было создано комплексное СПО, выполняющее все предъявляемые к нему требования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Министерство образования и науки Российской Федерации. Общие требования к учебным текстовым документам [Электронный ресурс]: СТО 02068410-004-2018 : стандарт организации : [принят 9 окт. 2007 г.] – 2018. – URL: http://repo.ssau.ru/handle/Metodicheskie-izdaniya/Obshie-trebovaniya-k-uchebnym-tekstovym-dokumentam-Elektronnyi-resurs-STO-020684100042018-standart-organizacii-prinyat-9-okt-2007-g-s-izm-utverzhd-v-fevr-2018-g-72605 (дата обращения: 29.04.2021).

2 Страница документации и учебных ресурсов Майкрософт для разработчиков и технических специалистов. / Microsoft – 2021. – URL: https://docs.microsoft.com/ru-ru/ (даты обращения: 01.03.2021 – 29.04.2021).

ПРИЛОЖЕНИЕ А
Руководство пользователя

1 Назначение программы

Системное программное обеспечение предназначено для выполнения следующих задач:

– выполнение синтаксической конструкции условного перехода;

– хранение, добавление, удаление и изменение записи о доступе пользователя;

– умножение двух целочисленных значений с проверкой переполнения.

2 Условия выполнения программы

Для работы синтаксической конструкции условного перехода необходимо соблюдать следующие условия:

1) Перед вводом конструкции условного перехода необходимо объявить 3 переменные типа double с именами: result, oper1, oper2;

2) Значения вводимых чисел не должны содержать символ «E»;

3) Поддерживаемые операции сравнения "<", ">", "=", ">=", "<=", "!=".

Для работы с информацией о пользователе необходимо соблюдать следующие условия:

1) Файл, хранящий информацию о пользователе, должен быть типа «binary»;

2) Хэш код пароля и логин пользователя должны быть уникальными.

Для работы с умножением двух целочисленных значений необходимо соблюдать следующие условия:

1) Вводимые числа должны быть целочисленными и не содержать дополнительных символов.

3 Выполнение программы

3.1 Стартовое окно

Предназначено для отображения сообщений о ходе выполняемой работы и возможности выбора необходимого для выполнения действия.

Начальное окно приложения (рисунок А.1) состоит из 4 подменю и графического элемента TextArea в котором происходит запись текущей работы (логирование):

1) Enter sequence – синтаксический анализ условного перехода if (<условие>) {<действие1> [else {<действие2>}]» и определения ветки в которой произойдет действие;
2) File Functionality – организация работы с binary файлом, данные которого отображаются в таблице;
3) Assembly – вычисления при помощи языка Assembler беззнакового целочисленного умножения;
4) Info – предоставление информации о создателе программы и возможностях, предоставляемых программой.

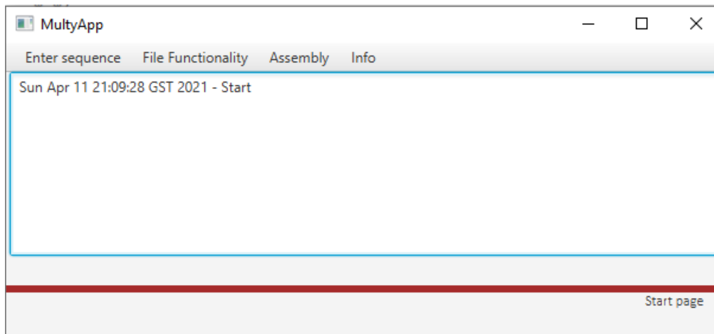

Рисунок А.1 – Начальное окно приложения

3.2 Окно синтаксической конструкции условного перехода

Предназначено для обеспечения выполнения конструкции условного перехода. Sequence checker состоит из (рисунок А.2):

1) Окно ввода кода;
2) Кнопка «Ok»;
3) Кнопка «Refresh»;
4) Подменю «Info».

Для начала работы необходимо ввести последовательность символов в объект TextArea.

Пример правильной последовательности:

```
double result;
double oper1 = 10;
double oper2 = 100;
if(oper1 > oper2)
{ result = Math.pow(oper1, 2);
} else { result = Math.pow(oper2, 3);}
```

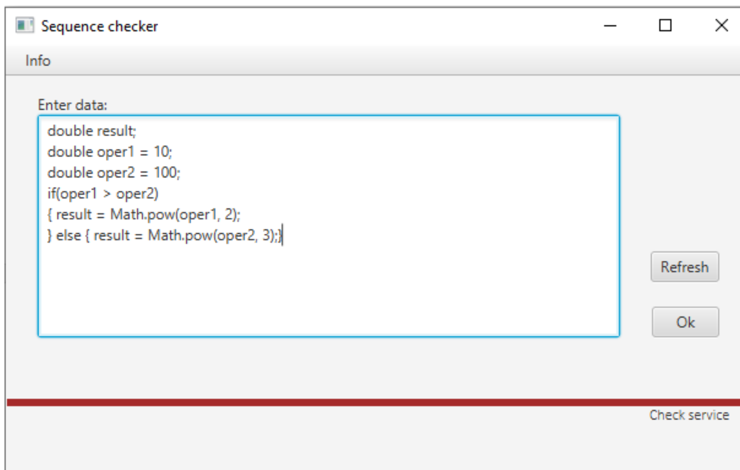

Рисунок А.2 - Sequence checker

Для начала выполнения введенного кода, нажмите на кнопку «Ok»:

Результат введенного кода представлен на рисунке А.3

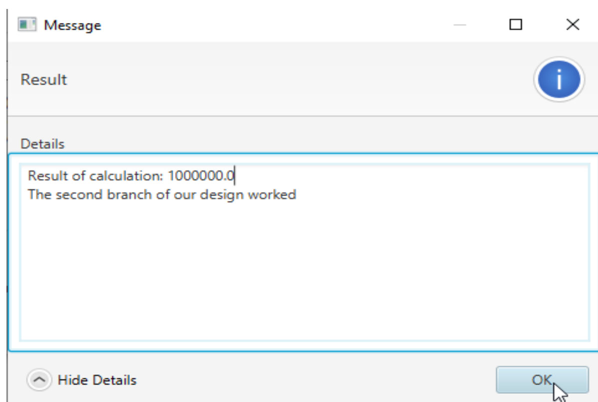

Рисунок А.3– Результат введенной последовательности

3.3 Сообщения оператору

В случае нарушения условий выполнения программы или грамматической ошибки, пользователь получит сообщение с информацией о своей ошибке.

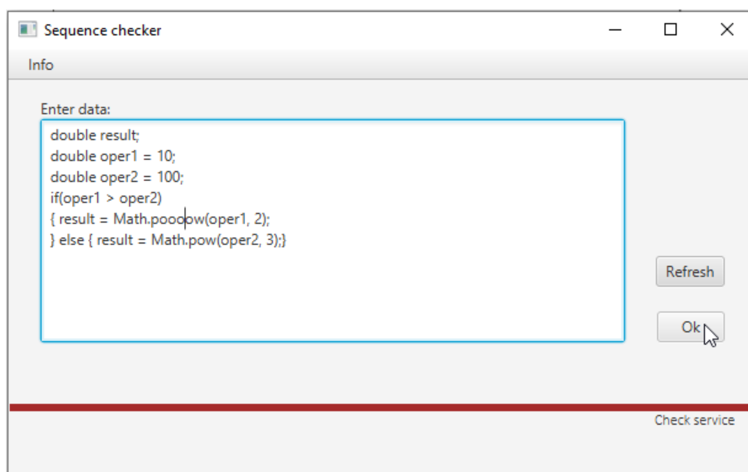

Рисунок А.4 – Неправильно введенная последовательность

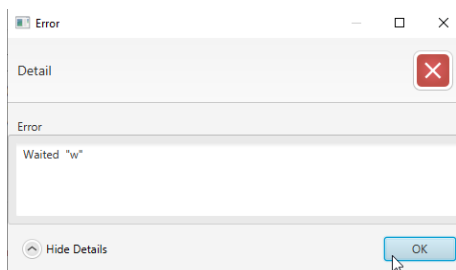

Рисунок А.5 – Результат неправильно введенной последовательности

3.4 Файловый менеджер

Предназначен для обеспечения операций хранения, добавления, удаления и изменения записи о доступе пользователя. Файловый менеджер состоит из(рисунокА.6):

1) Подменю «Info»;
2) Графического элемента TableView;
3) Кнопки «Add»;
4) Кнопки «Delete»;
5) Кнопки «Back»;

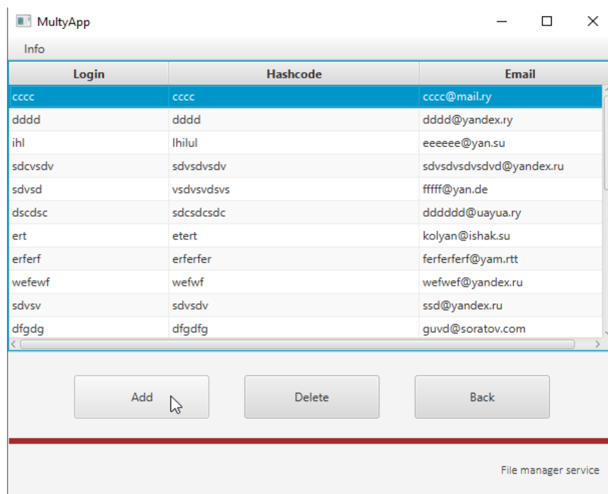

Рисунок А.6 – Менеджер записей

3.5 Операция добавления пользователя

Для выполнения операции добавления новой записи необходимо кликнуть на кнопку «Add». Произойдет открытие окна, в которое необходимо занести информацию о новом пользователе. Пример заполнения представлен на рисунке А.7.

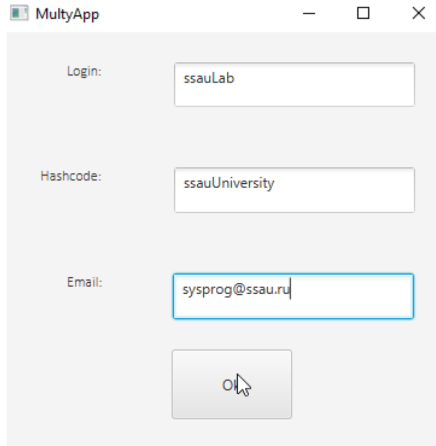

Рисунок А.7 – Окно ввода информации о новом пользователе

Для подтверждения введенной информации нажмите на кнопку «Ok». Произойдет переход на прошлую форму, где мы можем убедиться в успешном добавлении нашей записи.

3.6 Удаление записи о пользователе

Для удаления записи необходимо щелкнуть на необходимую запись в таблице, затем нажать кнопку «Delete». Выделенная запись исчезнет рисунок А.9.

3.7 Изменение данных о пользователе

Для изменения данных о пользователе необходимо кликнуть два раза по полю в записи, которое будет изменено. В редакторе, который откроется в ячейке, которую вы выбрали, введем новое значение рисунок А.10.

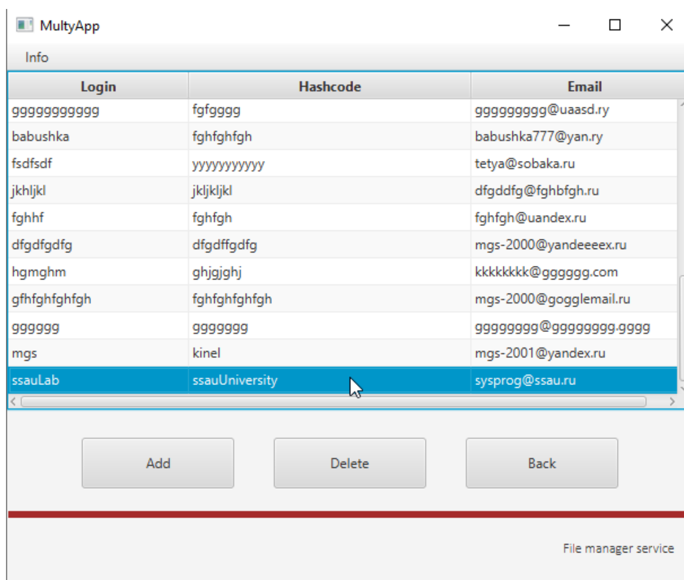

Рисунок А.8 – Запись добавлена в конец файла

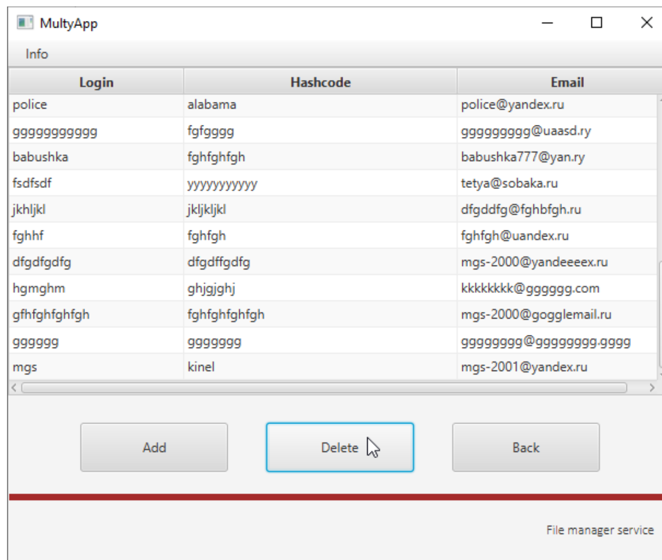

Рисунок А.9 – Удаление и авто обновление таблицы

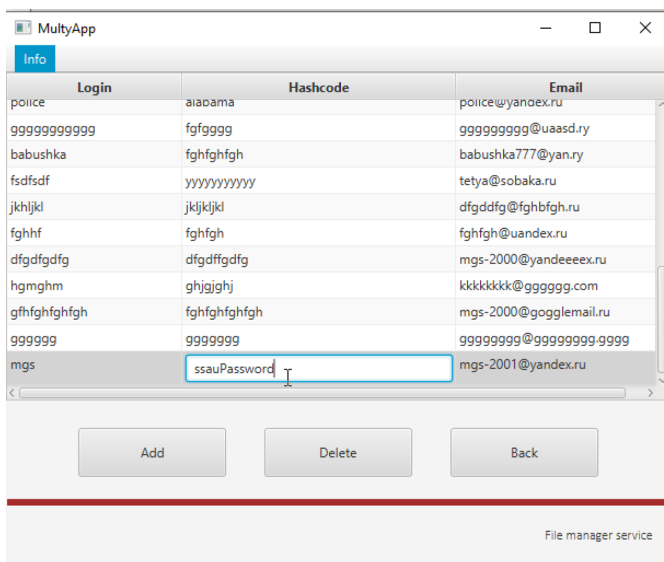

Рисунок А.10 – Открытие редактора для изменения одного поля таблицы

После ввода новой записи, нажмите клавишу «Enter». Авто обновление позволит вам сразу убедиться в корректности изменения значения рисунок А.11.

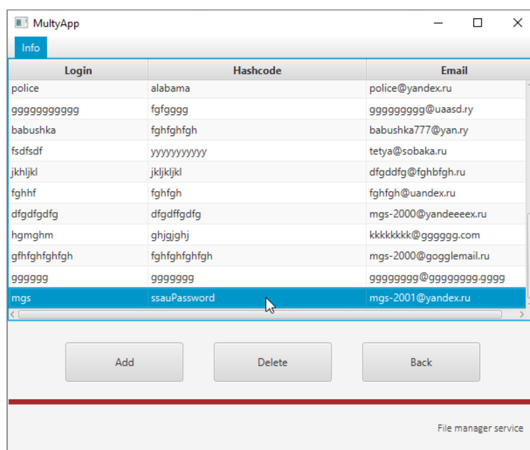

Рисунок А.11– Завершение редактирования и отображение обновленного значения

3.8 Сообщения оператору

В случае нарушения условий выполнения программы или грамматической ошибки, пользователь получит сообщение с информацией о своей ошибке рисунок А.13.

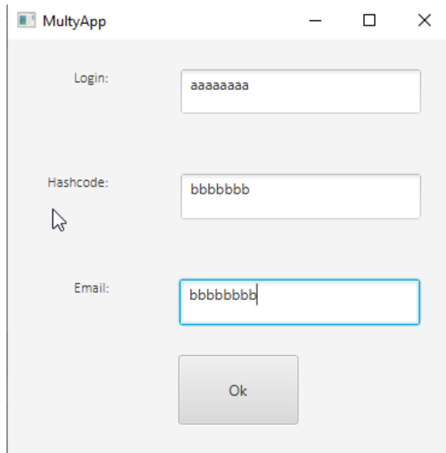

Рисунок А.12 – Ввод некорректного Email'a

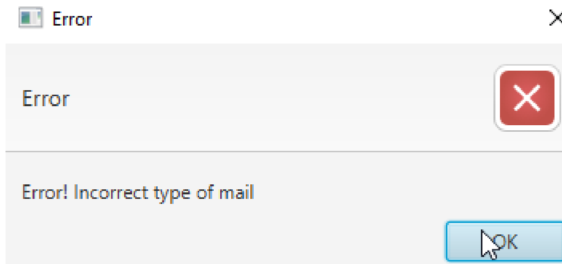

Рисунок А.13 – Полученное сообщение при неправильном вводе данных

3.9 Умножение двух целочисленных значений

Предназначен для обеспечения операции умножения двух целочисленных значений. Assembly состоит из (рисунок А.14):

1. Label и TextArea для ввода значения x;

2. Label и TextArea для ввода значения y;
3. Кнопка «Ok».

Для произведения операции умножения необходимо ввести два числа в соответствии с условиями выполнения программы.

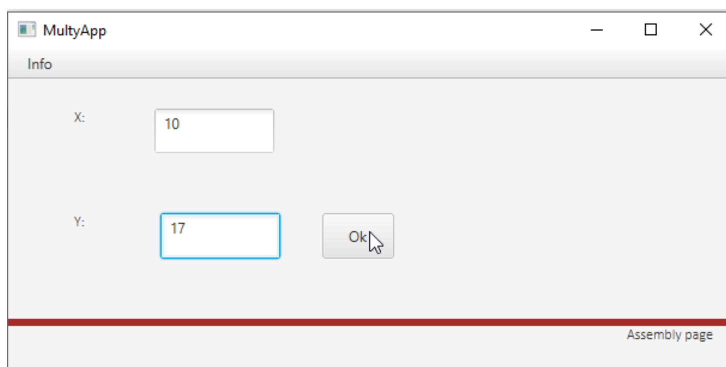

Рисунок А.14 – Окно Assembly

После заполнения двух полней кликнете на кнопку «Ok». Произойдет открытие окна с результатом умножения рисунок А.15.

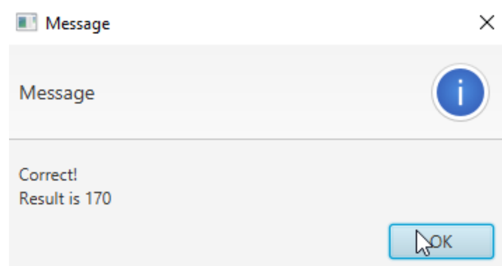

Рисунок А.15 – Результат выполнения умножения

3.10 Сообщения оператору

В случае нарушения условий выполнения программы или грамматической ошибки, пользователь получит сообщение с информацией о своей ошибке рисунок А.17.

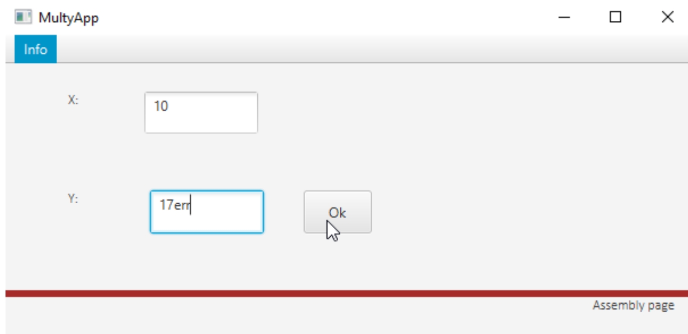

Рисунок А.16 – Ввод некорректных данных

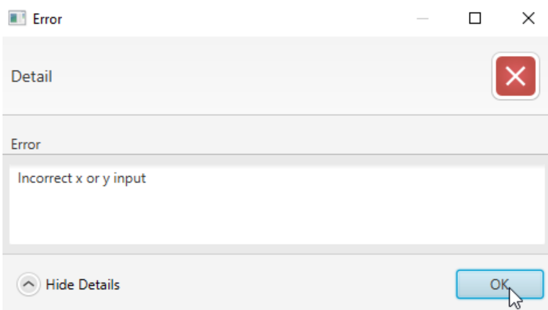

Рисунок А.17 – Результат выполнения умножения при неверных данных

Перечень принятых сообщений.

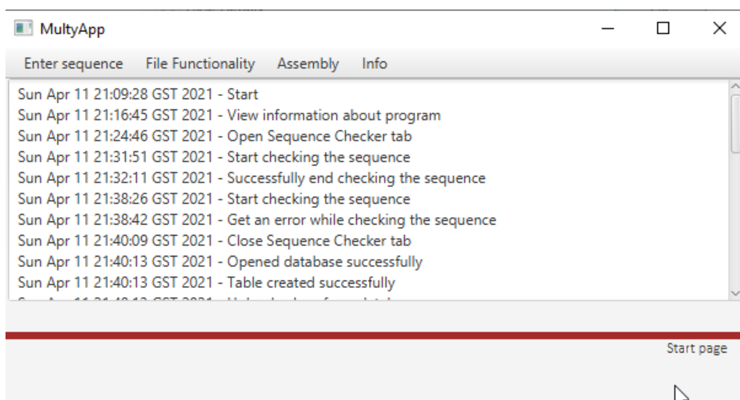

Рисунок А.18 – Окно логирования в начале выполнения программы

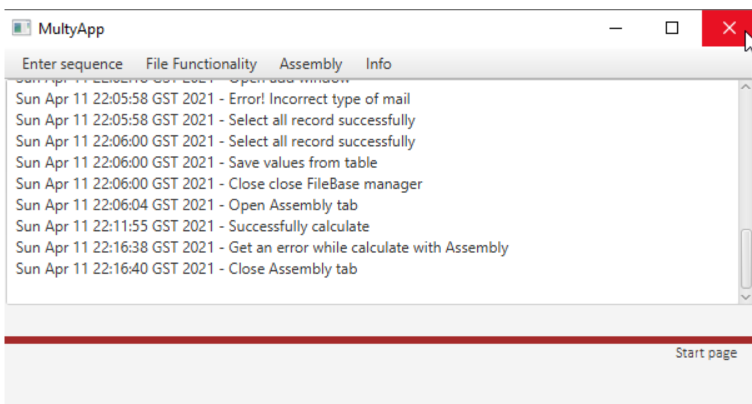

Рисунок В.19 – Окно логирования в конце выполнения программы

Учебное издание

Головнин Олег Константинович,
Столбовая Анастасия Александровна

ВВЕДЕНИЕ В СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ
И ОСНОВЫ ЖИЗНЕННОГО ЦИКЛА
СИСТЕМНЫХ ПРОГРАММ

Учебное пособие

Текст печатается в авторской редакции
Компьютерная верстка *А.С. Никитиной*

Подписано в печать 06.12.2021. Формат 60х84 1/16.
Бумага офсетная. Печ. л. 10,75.
Тираж 30 экз. Заказ .

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)
443086, САМАРА, МОСКОВСКОЕ ШОССЕ, 34.

Издательство Самарского университета.
443086, Самара, Московское шоссе, 34.