

УтверждАЮ

Зам. директора по ОУП

\_\_\_\_\_\_\_\_\_\_\_И.Г. Бозрова

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2025 г.

**ВОПРОСЫ К КОМПЛЕКСНОМУ ЭКЗАМЕНУ**

по междисциплинарным курсам:

МДК. 05.02. Разработка кода информационных систем

МДК. 05.03. Тестирование кода информационных систем

профессионального модуля:

ПМ.05. Проектирование и разработка информационных систем

специальность 09.02.07 Информационные системы и программирование (квалификация: специалист по информационным ресурсам)

курс 4

Мастер производственного обучения \_\_\_\_\_\_\_\_\_\_/В.В. Ледовских

Преподаватель \_\_\_\_\_\_\_\_\_\_/Т.В. Прилепская

*Теоретические вопросы по МДК. 05.03. Тестирование кода информационных систем*

### **1. Тестирование: понятие, критерии, отладка**

* **Тестирование** – проверка ПО на соответствие требованиям.
* **Критерии** – полнота покрытия, достоверность, автоматизируемость.
* **Процесс отладки**: выявление ошибки → анализ → исправление → повторное тестирование.

### **2. Цели и принципы тестирования**

* **Цели:** найти ошибки, проверить соответствие требованиям.
* **Принципы:** тестирование показывает наличие ошибок, раннее тестирование, тестирование зависит от контекста.

### **3. Стадии разработки ПО**

1. Анализ требований.
2. Проектирование.
3. Кодирование.
4. Тестирование.
5. Внедрение и поддержка.

### **4. Формирование тестовых наборов**

* **Эквивалентное разбиение** – тестирование по группам входных данных.
* **Граничные значения** – проверка минимальных и максимальных значений.
* **Негативное тестирование** – тестирование на некорректных данных.

### **5. Среда тестирования, тестовый план, отчет**

* **Среда тестирования** – инструменты и условия для тестов.
* **Тестовый план** – описание стратегии и методов тестирования.
* **Отчет о тестировании** – результаты тестов и найденные ошибки.

### **6. Методология тестирования сложных систем**

* **Сложная система** – большая, многокомпонентная.
* Методы тестирования: интеграционное, нагрузочное, стресс-тестирование.

### **7. Жизненный цикл ПО, стандарты**

* **Жизненный цикл ПО** – от разработки до завершения эксплуатации.
* **Стандарты тестирования:** ISO/IEC 9126 (качество), ISO/IEC 29119 (тестирование).

### **8. Каскадная модель жизненного цикла**

* Последовательные этапы разработки.
* Хорошо подходит для **стабильных проектов**.
* Минус – дорого исправлять ошибки на поздних стадиях.

### **9. Спиральная модель жизненного цикла**

* Разработка идет **циклами** (итерациями).
* Учитывает **риски**, но сложнее в управлении.

### **10. Каскадная vs Спиральная модель**

| **Характеристика** | **Каскадная модель (Waterfall)** | **Спиральная модель (Spiral)** |
| --- | --- | --- |
| **Структура** | Последовательные этапы | Итерации (циклы) разработки |
| **Гибкость** | Жесткая, изменения сложны | Гибкая, позволяет изменять требования |
| **Этапы** | Четкие фазы: анализ → проектирование → кодирование → тестирование → внедрение | В каждой итерации: анализ рисков → проектирование → разработка → тестирование |
| **Когда использовать?** | Если требования **четко определены** и не меняются | Если проект **сложный** и возможны изменения |
| **Риски** | Высокие – ошибки находят на поздних стадиях | Низкие – тестирование на каждом этапе |
| **Стоимость исправлений** | Дорого исправлять ошибки на поздних этапах | Дешевле, так как ошибки находят раньше |
| **Пример проекта** | Государственные системы, промышленные проекты | Долгосрочные, сложные проекты (банковские системы, ERP) |

✅ **Вывод:**

* **Каскадная модель** – для **простых и стабильных** проектов.
* **Спиральная модель** – для **сложных и изменяемых** проектов.

### **11. Характеристики ПО, алгоритмы**

### **Характеристики программного обеспечения (ПО)**

**1. Функциональность** – соответствует ли ПО требованиям.  
**2. Надежность** – стабильность работы без сбоев.  
**3. Производительность** – скорость работы при нагрузке.  
**4. Юзабилити** – удобство использования для пользователя.  
**5. Поддерживаемость** – легкость исправления ошибок и обновлений.  
**6. Безопасность** – защита данных от угроз.  
**7. Переносимость** – возможность работы на разных платформах.

### **Алгоритмы: определение и свойства**

**Алгоритм** – это последовательность шагов для решения задачи.

**Свойства алгоритма:**  
✅ **Дискретность** – алгоритм состоит из отдельных шагов.  
✅ **Определенность** – каждый шаг точно описан.  
✅ **Конечность** – выполнение завершается за конечное число шагов.  
✅ **Результативность** – получение ожидаемого результата.  
✅ **Массовость** – применим для разных входных данных.

### **12. Виды тестирования**

* **Функциональное** – проверка работы функций.
* **Нефункциональное** – производительность, безопасность.
* **Модульное** – проверка отдельных компонентов.
* **Интеграционное** – проверка взаимодействия модулей.

### **13. Функциональное тестирование**

* Проверяет **что делает система**.
* Пример: проверка кнопки "Вход" в форму авторизации.

### **14. Нефункциональное тестирование**

* Проверяет **как работает система** (скорость, нагрузка, безопасность).
* Пример: выдержит ли сайт 100 000 пользователей.

### **15. Модульное тестирование**

* Проверка **отдельных частей кода**.
* **Методы**: Assert-тесты, тестирование граничных значений.
* **Плюсы:** ошибки выявляются рано.
* **Минусы:** не тестирует взаимодействие модулей.

### **Дымовое vs Санитарное тестирование: различия**

| **Характеристика** | **Дымовое тестирование (Smoke Testing)** | **Санитарное тестирование (Sanity Testing)** |
| --- | --- | --- |
| **Цель** | Проверить, работает ли система в целом | Проверить, исправлены ли конкретные баги |
| **Область тестирования** | Все основные функции системы | Только измененные компоненты |
| **Когда применяется?** | На ранних этапах тестирования, перед детальной проверкой | После внесения исправлений, перед регрессионным тестированием |
| **Глубина проверки** | Поверхностная, основные функции | Глубже, но только для исправленных частей |
| **Пример** | Проверяем, загружается ли приложение, можно ли войти в систему, открыть меню | Проверяем, исправлена ли ошибка с кнопкой «Отправить» |
| **Результат** | Если не пройдено – тестирование прекращается | Если не пройдено – возвращаем разработчикам на доработку |

✅ **Вывод:**

* **Дымовое тестирование** – проверяет **основную работоспособность ПО**.
* **Санитарное тестирование** – проверяет, **правильно ли исправлены баги**.

### **18. Интеграционное тестирование**

* **Определение**: Проверка взаимодействия модулей системы.
* **Необходимость**: Выявление проблем при передаче данных между модулями.
* **Методы**:
  + **Сверху вниз** – тестирование верхних уровней системы.
  + **Снизу вверх** – сначала тестируются низкоуровневые компоненты.
  + **Большой взрыв** – тестируются все модули одновременно.
* **Критерии входа**: Модули готовы, код протестирован отдельно.
* **Критерии выхода**: Ошибки интеграции устранены, данные передаются корректно.

### **19. Тестирование методом «черного ящика»**

* **Определение**: Тестирование без знания кода.
* **Типы**: Функциональное, регрессионное, нагрузочное тестирование.
* **Область**: Пользовательский интерфейс, API, базы данных.
* **Примеры**: Проверка входа в систему без анализа кода.

### **20. Тестирование методом «белого ящика»**

* **Определение**: Тестирование с анализом внутреннего кода.
* **Типы**: Модульное, интеграционное тестирование.
* **Область**: Проверка алгоритмов, логики кода.
* **Преимущества**: Высокая точность выявления ошибок.
* **Недостатки**: Требует знаний программирования.
* **Пример**: Проверка корректности выполнения циклов и условий.

### **21. Тестирование методом «серого ящика»**

* **Стратегия**: Комбинация тестирования «черного» и «белого» ящика.
* **Преимущества**: Оптимальный баланс между знанием кода и тестами интерфейса.
* **Проблемы**: Сложность анализа кода и сценариев тестирования.
* **Область**: API, базы данных, взаимодействие компонентов.
* **Этапы**: Анализ требований → Выбор тестов → Проверка поведения системы.

### **22. Регрессивное тестирование**

* **Определение**: Проверка, не появились ли новые ошибки после внесения изменений.
* **Методы**: Полное, частичное регрессионное тестирование.
* **Выбор тестов**: Критичные функции, затронутые изменения.
* **Различия с повторным тестированием**:
  + **Повторное тестирование** – проверяет конкретные исправления.
  + **Регрессионное тестирование** – проверяет, не сломалось ли что-то еще.
* **Проблемы**: Увеличение времени тестирования.

### **23. Ручное тестирование**

* **Определение**: Выполнение тестов без автоматизации.
* **Преимущества**: Гибкость, возможность проверки UI.
* **Недостатки**: Медленное, подвержено человеческим ошибкам.
* **Типы**: Функциональное, UX-тестирование, исследовательское тестирование.

### **24. Юзабилити-тестирование**

* **Определение**: Проверка удобства использования интерфейса.
* **Подходы**: Анализ пользовательского опыта, A/B-тестирование.
* **Пример**: Проверка удобства навигации по сайту.

### **25. Экспертный подход к юзабилити-тестированию**

Анализ удобства интерфейса **экспертами**, без реальных пользователей. Оценка на основе **лучших UX/UI практик**.

📌 **Критерии:**  
✔ **Эффективность** – удобство выполнения задач.  
✔ **Простота** – минимизация лишних действий.  
✔ **Консистентность** – единый стиль.  
✔ **Обратная связь** – понятность ошибок и подсказок.  
✔ **Адаптивность** – корректное отображение на разных устройствах.

📌 **Методы:**  
🔹 **Эвристический анализ** – проверка по правилам Нильсена.  
🔹 **Когнитивное прохождение** – эксперт анализирует сценарии.  
🔹 **Анализ доступности** – удобство для пользователей с ограниченными возможностями.

📌 **Плюсы:**  
✅ Быстро, не требует пользователей.  
✅ Основано на UX-стандартах.

📌 **Минусы:**  
❌ Возможна субъективность.  
❌ Не учитывает реальные сценарии пользователей.

📌 **Вывод:**  
Используется **на ранних этапах** для выявления UX-проблем, но **не заменяет тестирование реальными пользователями**. 🚀

### **26. Пользовательский подход к юзабилити-тестированию**

* **Критерии**: Реальная работа пользователей с интерфейсом.
* **Методы**: Тестирование с наблюдением, опросы.
* **Преимущества**: Тестирование в реальных условиях.
* **Недостатки**: Долго, дорого.

### **27. Автоматизация тестирования**

* **Определение**: Использование программ для автоматического тестирования.
* **Типы**: UI-тестирование, API-тестирование, нагрузочное тестирование.
* **Принципы**: Надежность, повторяемость.
* **Требования к тестировщику**: Знание языков программирования и инструментов тестирования.
* **Преимущества**: Быстрое выполнение тестов.
* **Недостатки**: Долгая настройка, высокая стоимость.

### **28. Утверждения (Assertions)**

* **Определение**: Проверки в автоматизированных тестах.
* **Примеры**:
  + Assert.AreEqual(5, GetSum(2,3)) – проверка суммы.
  + Assert.True(IsValidEmail("test@mail.com")) – проверка email.

### **29. Тестирование производительности**

* **Определение**: Проверка скорости работы системы.
* **Этапы**: Определение метрик → Запуск тестов → Анализ.
* **Критерии**: Время отклика, количество запросов в секунду.

1. *Теоретические вопросы по МДК. 05.02. Разработка кода информационных систем*

### **1. Структура и виды CASE-средств. Критерии при выборе.**

* **CASE-средства** – инструменты автоматизации разработки ПО.
* **Виды**:
  + **Upper CASE** – для проектирования (UML, ER-диаграммы).
  + **Lower CASE** – для кодирования и тестирования (Eclipse, Visual Studio).
  + **Integrated CASE** – объединяет верхний и нижний уровни.
* **Критерии выбора**: функциональность, поддержка стандартов, интеграция с другими инструментами.

### **2. Структуры среды разработки. Факторы при выборе.**

* **Структуры**:
  + **Локальная** – работает на одном компьютере.
  + **Клиент-серверная** – используется в командах (например, Git + CI/CD).
  + **Облачная** – удаленный доступ через интернет (GitHub Codespaces).
* **Факторы выбора**: сложность проекта, количество разработчиков, поддержка языков программирования.

### **3. Характеристика основных возможностей среды разработки.**

* **Редактор кода** (подсветка синтаксиса, автодополнение).
* **Отладчик** (постановка точек останова, анализ выполнения).
* **Компилятор/интерпретатор** (перевод кода в исполняемый файл).
* **Интеграция с системой контроля версий** (Git).

### **4. Основные инструменты среды для создания, исполнения и управления информационной системой.**

* **Создание** – текстовые редакторы, шаблоны проектов.
* **Исполнение** – компиляторы, интерпретаторы, эмуляторы.
* **Управление** – CI/CD, Git, базы данных, профилирование кода.

### **5. Средства обработки информации: определение, виды.**

* **Определение**: инструменты для хранения, анализа и преобразования данных.
* **Виды**:
  + **Физические** (серверы, хранилища данных).
  + **Программные** (SQL, Big Data, AI-аналитика).

### **6. Организация работы в команде разработчиков. Виды моделей и роли.**

* **Модели**:
  + **Каскадная** – поэтапное выполнение.
  + **Agile** – итерационное развитие продукта.
* **Роли**:
  + **Разработчик** – пишет код.
  + **Аналитик** – изучает требования.
  + **Тестировщик** – проверяет качество.

### **7. Система контроля версий: совместимость, установка, настройка.**

* **Совместимость** – поддержка Git, SVN, Mercurial.
* **Установка** – скачивание и настройка Git, создание репозитория.
* **Настройка** – конфигурация .gitignore, настройка CI/CD.

### **8. Характеристика методов обеспечения кроссплатформенности информационной системы.**

* **Методы**:
  + Использование **интерпретируемых языков** (Python, JavaScript).
  + **Контейнеризация** (Docker).
  + **Фреймворки** (Electron, Xamarin).

### **9. Особенности сервисно-ориентированной архитектуры (SOA), основные принципы.**

* **SOA** – архитектура, где функционал разделен на независимые сервисы.
* **Принципы**:
  + **Легковесность** – сервисы минимальны.
  + **Модульность** – можно заменить один сервис без влияния на другие.
  + **Гибкость** – поддержка множества клиентов (REST, SOAP).

### **10. Интегрированные среды разработки для создания независимых программ: назначение, сфера применения.**

* **Назначение**: Упрощение разработки за счет инструментов (IDE).
* **Сферы**: Web-разработка, мобильные приложения, серверное ПО.

### **11. Особенности объектно-ориентированных языков программирования.**

* **Основные концепции**:
  + **Инкапсуляция** – скрытие внутренней логики.
  + **Наследование** – переиспользование кода.
  + **Полиморфизм** – один интерфейс, много реализаций.

### **12. Диаграммы UML: варианты использования, последовательности, кооперации.**

* **Варианты использования** – отображают взаимодействие пользователя с системой.
* **Последовательности** – показывают порядок вызова методов.
* **Кооперации** – связи между объектами.

### **13. Диаграммы UML: развертывания, компонентов, потоков данных.**

* **Развертывания** – показывает размещение компонентов на серверах.
* **Компонентов** – связи между модулями системы.
* **Потоков данных** – движение данных в системе.

### **14. Архитектура информационной системы: виды, типы групп, применение архитектурных описаний.**

* **Виды архитектур**:
  + **Многослойная** (клиент-сервер).
  + **Микросервисная** (независимые сервисы).
* **Применение описаний** – проектирование, документация.

### **15. Определение конфигурации информационной системы. Этапы, методы и средства конфигурирования.**

* **Конфигурация** – настройки, параметры системы.
* **Этапы**: анализ → настройка → тестирование.
* **Методы**: ручное и автоматическое конфигурирование.
* **Средства**: Ansible, Docker, CI/CD.

### **16. Критерии выбора технических средств проекта информационной системы**

* **Производительность** – соответствие мощности оборудования задачам.
* **Совместимость** – поддержка операционных систем, БД.
* **Масштабируемость** – возможность расширения.
* **Безопасность** – защита данных и отказоустойчивость.

### **17. Репозиторий проекта информационной системы: понятие и процесс создания**

* **Репозиторий** – централизованное хранилище кода и документации.
* **Процесс создания**:
  1. **Инициализация** (git init).
  2. **Добавление файлов** (git add).
  3. **Создание коммита** (git commit -m "Init").
  4. **Подключение к удаленному серверу** (git remote add origin).
  5. **Отправка в репозиторий** (git push).

### **18. Алгоритм определения уровня доступа к репозиторию в системе контроля версий**

1. **Определение ролей пользователей** (разработчик, тестировщик, администратор).
2. **Настройка прав** (read, write, admin).
3. **Настройка доступа в системе контроля версий** (GitHub, GitLab, Bitbucket).
4. **Регулярный аудит доступа** (проверка логов).

### **19. Процесс настройки среды разработки информационной системы**

* **Выбор среды** (VS Code, IntelliJ IDEA, PyCharm).
* **Установка интерпретаторов и компиляторов** (JDK, Python, .NET).
* **Настройка зависимостей** (npm install, pip install).
* **Настройка системы контроля версий** (git config --global user.name).
* **Интеграция с CI/CD** (Jenkins, GitHub Actions).

### **20. Инструменты мониторинга разработки проекта**

* **Для отслеживания задач** – Jira, Trello.
* **Для логирования** – ELK Stack, Logstash.
* **Для мониторинга серверов** – Prometheus, Grafana.
* **Для анализа кода** – SonarQube, ESLint.

### **21. Требования к пользовательскому интерфейсу информационной системы**

* **Юзабилити** – удобство использования.
* **Кроссплатформенность** – работа на разных устройствах.
* **Адаптивность** – корректное отображение на экранах разного размера.
* **Доступность** – поддержка пользователей с ограниченными возможностями.

### **22. Принципы создания графического пользовательского интерфейса (GUI)**

* **Консистентность** – одинаковый стиль элементов.
* **Минимализм** – только необходимые элементы.
* **Обратная связь** – реакции интерфейса на действия пользователя.
* **Доступность** – контрастность, шрифты, голосовые подсказки.

### **23. Понятие и формы спецификации языка программирования**

* **Синтаксис** – правила записи кода.
* **Семантика** – смысл конструкций языка.
* **Примеры спецификаций**:
  + **ISO/IEC 9899** (C).
  + **ECMA-262** (JavaScript).
  + **Python PEP 8** (стиль кода).

### **24. Методы организации ввода-вывода информации в информационной системе**

* **Консольный ввод-вывод** (Console.ReadLine(), print() в Python).
* **Графический ввод-вывод** (GUI, Web-формы).
* **Файловый ввод-вывод** (File.ReadAllText(), open("file.txt")).
* **Сетевой ввод-вывод** (API, WebSocket).

### **25. Спецификация настроек типовой информационной системы**

* **Настройки безопасности** – шифрование, авторизация.
* **Локализация** – поддержка нескольких языков.
* **Конфигурация серверов** – базы данных, балансировщики нагрузки.
* **Параметры производительности** – кэширование, лимиты памяти.

### **26. Сообщения между модулями: назначение, вход и разделы модуля**

* **Назначение** – передача данных между частями системы.
* **Входные данные** – параметры API, файлы, события.
* **Разделы модуля**:
  + **Обработчик входных данных**.
  + **Логика обработки**.
  + **Выходные данные** (JSON, XML, текст).

### **27. Приложения для моделирования процессов: виды моделей, требования**

* **Виды моделей**:
  + **Диаграммы UML** (Use Case, Sequence, ER-диаграммы).
  + **Бизнес-процессы (BPMN)**.
* **Требования**:
  + Гибкость, читаемость, поддержка стандартов.

### **28. Разработка приложений: виды, этапы разработки**

* **Виды**:
  + **Web** – сайты, веб-приложения.
  + **Desktop** – программы для ПК.
  + **Mobile** – Android/iOS.
* **Этапы**:
  + Анализ требований.
  + Дизайн интерфейса.
  + Разработка кода.
  + Тестирование.
  + Внедрение.

### **29. Отладка приложения: определение, методы**

* **Определение**: поиск и исправление ошибок в коде.
* **Методы**:
  + **Логирование** (Console.WriteLine(), print()).
  + **Отладчик** (breakpoints, watch-variables).
  + **Unit-тестирование** (Assert.Equals(expected, actual)).

### **30. Интеграция модуля: определение, способы интеграции**

* **Определение**: объединение модуля в систему.
* **Способы интеграции**:
  + **API** (REST, SOAP).
  + **Базы данных** (SQL-запросы).
  + **MQ-сообщения** (RabbitMQ, Kafka).

Практические задания:

1. Напишите программу, в которой генерируется случайное целое число (например, в диапазоне от 1 до 10), а пользователю необходимо его угадать. Если пользователь не угадал число, программа выдает запрос о том, хочет ли он попробовать еще раз.

static void Main(String[] args)

{

Random rnd = new Random();

int number = rnd.Next(1, 11);

int guess;

Console.WriteLine("Угадай число от 1 до 10!");

do

{

Console.Write("Ваш вариант: ");

guess = int.Parse(Console.ReadLine());

if (guess == number)

{

Console.WriteLine("111");

break;

}

else

{

Console.WriteLine("Неправильно! Попробуете еще раз? (да/нет)");

}

} while (Console.ReadLine().ToLower() == "да");

Console.WriteLine($"Загаданное число: {number}");

}

2. Напишите программу, в которой пользователь вводит дату своего рождения, а программа вычисляет, сколько прошло полных лет, месяцев и дней от указанной даты до текущей.

static void Main(String[] args)

{

Console.Write("Введите дату рождения (в формате ГГГГ-ММ-ДД): ");

try

{

DateTime birtday = DateTime.Parse(Console.ReadLine());

DateTime today = DateTime.Today;

int years = today.Year - birtday.Year;

int months = today.Month - birtday.Month;

int day = today.Day - birtday.Day;

if (day < 0)

{

months--;

day += DateTime.DaysInMonth(today.Year, today.Month == 1 ? 12 : today.Month - 1);

}

if (months < 0) {

years--;

months += 12;

}

Console.WriteLine($"Вы прожили: {years} лет, {months} месяцев, {day} дней.");

}

catch

{

Console.WriteLine("Ошибка! Введите корректную дату в формате ГГГГ-ММ-ДД.");

}

}

3. Напишите программу, в которой считывается содержимое текстового файла и создается новый текстовый файл. В новый текстовый файл заносится текст из исходного текстового файла, но все пробелы заменяются подчеркиваниями, а заглавные буквы заменяются строчными.

Console.Write("Введите путь к исходному файлу: ");

string inputFilePath = Console.ReadLine();

Console.Write("Введите путь к новому файлу: ");

string outputFilePath = Console.ReadLine();

try

{

if (!File.Exists(inputFilePath))

{

Console.WriteLine("Ошибка: исходный файл не найден.");

return;

}

string content = File.ReadAllText(inputFilePath);

string modifiedContent = content.Replace(" ", "\_").ToLower();

File.WriteAllText(outputFilePath, modifiedContent);

Console.WriteLine("Файл успешно обработан и сохранён.");

}

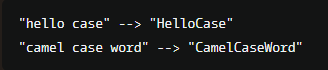
catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

5. Напишите метод, который преобразует строку в формат camelCase, то есть первая буква во всех словах должна быть заглавной, а пробелы должны быть удалены. Возможность сохранять данные в файл и считывать данные из файла. Пример изображен на рисунке.



static void Main()

{

Console.Write("Введите строку: ");

string input = Console.ReadLine();

string result = ToCamelCase(input);

Console.WriteLine("Результат: " + result);

}

static string ToCamelCase(string input)

{

string[] words = input.Split(' ');

for (int i = 0; i < words.Length; i++)

{

if (words[i].Length > 0)

words[i] = char.ToUpper(words[i][0]) + words[i].Substring(1).ToLower();

}

return string.Concat(words);

}

7. Напишите функцию, которая принимает строку из одного или нескольких слов и возвращает ту же строку, но все слова, содержащие пять или более букв, меняются местами (как в показано в примере). Передаваемые строки будут состоять только из букв и пробелов. Пробелы будут включены только в том случае, если присутствует более одного слова.

НЕ могу

8. Напишите функцию, которая принимает массив из 10 целых чисел (от 0 до 9) и возвращает строку из этих чисел в виде телефонного номера.

static void Main()

{

Console.Write("Введите 10 цифр: ");

string input = Console.ReadLine();

if (input.Length == 10 && long.TryParse(input, out \_))

Console.WriteLine($"({input[..3]}) {input[3..6]}-{input[6..]}");

else

Console.WriteLine("Ошибка: введите 10 цифр.");

}

9. Напишите функцию, которая может принимать любое неотрицательное целое число в качестве аргумента и возвращать его с его цифрами в порядке убывания. По сути, переставьте цифры, чтобы получить максимально возможное число. Пример показан на рисунке.

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

static void Main()

{

Console.Write("Введите число: ");

string input = Console.ReadLine();

if (!long.TryParse(input, out \_))

{

Console.WriteLine("Ошибка: введите неотрицательное целое число.");

return;

}

char[] digits = input.ToCharArray();

Array.Sort(digits);

Array.Reverse(digits);

Console.WriteLine("Результат: " + new string(digits));

}

11. Создайте тестовый проект на MSUnit. Напишите тест для метода Add(int a, int b), в проекте bilet\_n, который возвращает сумму двух чисел. Убедитесь, что тест проверяет корректность работы метода для положительных, отрицательных и нулевых значений.

public class Calculator

{

public static int Add(int a, int b)

{

return a + b;

}

}

[TestClass]

public class CalculatorTests

{

[TestMethod]

public void Add\_PositiveNumbers\_ReturnsCorrectSum()

{

int result = Calculator.Add(5, 7);

Assert.AreEqual(12, result);

}

[TestMethod]

public void Add\_NegativeNumbers\_ReturnsCorrectSum()

{

int result = Calculator.Add(-3, -8);

Assert.AreEqual(-11, result);

}

}

12. Создайте тестовый проект на NUnit. Напишите тест для метода Divide(int a, int b) в проекте bilet\_n, который выбрасывает исключение DivideByZeroException, если b равен нулю. Проверьте, что исключение действительно выбрасывается.

public class Calculator

{

public static int Divide(int a, int b)

{

if (b == 0)

throw new DivideByZeroException("Деление на ноль невозможно!");

return a / b;

}

}

[TestFixture]

public class CalculatorTests

{

[Test]

public void Divide\_ByNonZero\_ReturnsCorrectResult()

{

int result = Calculator.Divide(10, 2);

Assert.AreEqual(5, result);

}

[Test]

public void Divide\_ByZero\_ThrowsDivideByZeroException()

{

Assert.Throws<DivideByZeroException>(() => Calculator.Divide(10, 0));

}

}

13. Создайте тестовый проект на MSUnit. Создайте параметризованный тест для метода IsEven(int number) проекта bilet\_n, который проверяет, является ли число четным. Используйте атрибут [DataTestMethod] и [DataRow] для передачи различных входных данных.

public class NumberUtils

{

public static bool IsEven(int number)

{

return number % 2 == 0;

}

}

// MSUnit

[TestClass]

public class NumberCheckerTests

{

[DataTestMethod]

[DataRow(2, true)]

[DataRow(3, false)]

[DataRow(0, true)]

[DataRow(-4, true)]

[DataRow(-7, false)]

public void IsEven\_TestCases(int number, bool expected)

{

bool result = NumberUtils.IsEven(number);

Assert.AreEqual(expected, result);

}

}

// XUnit

public class Test

{

[Theory]

[InlineData(2, true)]

public void Test1(int n, bool v)

{

Assert.Equal(NumberUtils.IsEven(n), v);

}

}

//Nunit

[TestFixture]

public class test

{

[TestCase(2, true)]

[TestCase(3, false)]

[TestCase(0, true)]

[TestCase(-4, true)]

[TestCase(-7, false)]

public void Test(int n, bool v)

{

Assert.AreEqual(NumberUtils.IsEven(n), v);

}

}

14. Создайте тестовый проект на NUnit. Напишите тест для метода Factorial(int n) проекта bilet\_n, который вычисляет факториал числа. Проверьте корректность работы метода для различных входных данных, включая граничные случаи.

public class MathUtils

{

public static long Factorial(int n)

{

if (n < 0)

throw new ArgumentException("Число должно быть неотрицательным!");

long result = 1;

for (int i = 2; i <= n; i++)

{

result \*= i;

}

return result;

}

}

[TestFixture]

public class MathUtilsTests

{

[TestCase(0, 1)]

[TestCase(1, 1)]

[TestCase(2, 2)]

[TestCase(3, 6)]

[TestCase(5, 120)]

[TestCase(10, 3628800)]

public void Factorial\_ValidNumbers(int n, long expected)

{

Assert.AreEqual(expected, MathUtils.Factorial(n));

}

[Test]

public void Factorial\_NegativeNumber\_ThrowsException()

{

Assert.Throws<ArgumentException>(() => MathUtils.Factorial(-1));

}

}

15. Создайте тестовый проект на xUnit. Напишите тест для метода IsPalindrome(string text) проекта bilet\_n, который проверяет, является ли строка палиндромом. Проверьте корректность работы метода для различных строк.

public class Palindrome

{

public static bool IsPalindrome(string text)

{

char[] t = text.ToLower().Where(char.IsLetterOrDigit).ToArray();

return t.SequenceEqual(t.Reverse());

}

}

public class PalindromeTests

{

[Theory]

[InlineData("racecar", true)]

[InlineData("level", true)]

[InlineData("Madam", true)]

[InlineData("A Santa at NASA", true)]

[InlineData("hello", false)]

[InlineData("12321", true)]

public void IsPalindrome\_TestCases(string input, bool expected)

{

bool result = Palindrome.IsPalindrome(input);

Assert.Equal(expected, result);

}

}

16. Напишите тестовые случаи для функции, которая проверяет, является ли строка валидным email-адресом. Функция возвращает true, если email валиден, и false в противном случае.

public class EmailValidator

{

public static bool IsValidEmail(string email)

{

if (string.IsNullOrWhiteSpace(email) || !email.Contains("@"))

return false;

string[] parts = email.Split('@');

if (parts.Length != 2 || parts[0].Length == 0 || parts[1].Length < 3 || !parts[1].Contains("."))

return false;

return true;

}

}

public class EmailValidatorTests

{

[Theory]

[InlineData("test@example.com", true)]

[InlineData("valid.email@sub.com", true)]

[InlineData("user@gmail.com", true)]

[InlineData("invalid@domain", false)] // Нет ".com"

[InlineData("@missingusername.com", false)] // Нет имени

[InlineData("plainaddress", false)] // Нет "@"

[InlineData("", false)] // Пустая строка

public void IsValidEmail\_TestCases(string email, bool expected)

{

bool result = EmailValidator.IsValidEmail(email);

Assert.Equal(expected, result);

}

}

17. Напишите тестовые случаи для функции, которая принимает сумму покупки и возвращает размер скидки в зависимости от суммы:

a. Если сумма меньше 1000, скидка 0%.

b. Если сумма от 1000 до 5000, скидка 5%.

c. Если сумма больше 5000, скидка 10%.

public class DiscountCalculator

{

public static decimal GetDiscount(decimal amount)

{

if (amount < 1000) return 0;

if (amount >= 1000 && amount <= 5000) return 5;

return 10;

}

}

18. Напишите тестовые случаи для функции, которая принимает массив чисел и возвращает максимальное число. Если массив пуст, функция возвращает null.

public static int? GetMaxNumber(int[] numbers)

{

if (numbers == null || numbers.Length == 0)

return null;

return numbers.Max();

}

19. Напишите тестовые случаи для функции, которая проверяет, соответствует ли пароль следующим требованиям:

a. Длина пароля не менее 8 символов.

b. Пароль содержит хотя бы одну цифру.

c. Пароль содержит хотя бы одну заглавную букву.

d. Пароль содержит хотя бы один специальный символ (!@#$%^&\*).

Функция возвращает true, если пароль валиден, и false в противном случае.

public class PasswordValidator

{

public static bool IsValidPassword(string password)

{

return password.Length >= 8 &&

password.Any(char.IsDigit) &&

password.Any(char.IsUpper) &&

password.Any("!@#$%^&\*".Contains);

}

}

public class PasswordValidatorTests

{

[Theory]

[InlineData("abcdefg1!", false)]

[InlineData("Abcdefg!!", false)]

[InlineData("Abc12345", false)]

[InlineData("A1!", false)]

[InlineData("12345678!", false)]

[InlineData("", false)]

public void IsValidPassword\_TestCases(string password, bool expected)

{

Assert.Equal(expected, PasswordValidator.IsValidPassword(password));

}

}

20. Напишите тестовые случаи для функции, которая принимает температуру в градусах Цельсия и конвертирует её в градусы Фаренгейта по формуле:

F = (C \* 9/5) + 32.

public class TemperatureConverter

{

public static double ConvertToFahrenheit(double celsius)

{

return (celsius \* 9 / 5) + 32;

}

}

public class TemperatureConverterTests

{

[Theory]

[InlineData(0, 32)]

[InlineData(100, 212)]

[InlineData(-40, -40)]

[InlineData(37, 98.6)]

[InlineData(-10, 14)]

[InlineData(50, 122)]

public void ConvertToFahrenheit\_TestCases(double celsius, double expectedFahrenheit)

{

double result = TemperatureConverter.ConvertToFahrenheit(celsius);

Assert.Equal(expectedFahrenheit, result, 1);

}

}

21. Протестируйте программу bilet\_n методом черного ящика, опишите тестовый сценарий.

22. Протестируйте программу bilet\_n методом черного ящика, опишите тестовый сценарий.

23. Протестируйте программу bilet\_n методом черного ящика, опишите тестовый сценарий.

24. Протестируйте программу bilet\_n методом черного ящика, опишите тестовый сценарий.

25. Протестируйте программу bilet\_n методом черного ящика, опишите тестовый сценарий.

Рассмотрено на заседании ПЦК ОПД и ПМ специальности 09.02.07. ИСП (специалист по информационным ресурсам, специалист по информационным системам, программист)

Протокол от «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г. № \_\_\_\_\_

Председатель ПЦК \_\_\_\_\_\_\_\_\_\_\_/Ю.В. Климова

Методист ОП№5 \_\_\_\_\_\_\_\_\_\_\_/Н.А. Есауленко

### **Шпаргалка по базовым командам NUnit, MSTest и xUnit**

| **Фреймворк** | **Атрибут для тестового класса** | **Атрибут для теста** | **Параметризованный тест** | **Запуск перед тестами** | **Запуск после тестов** | **Проверка исключения** | **Базовые проверки** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **MSTest** | [TestClass] | [TestMethod] | [DataTestMethod] + [DataRow(...)] | [TestInitialize] | [TestCleanup] | Assert.ThrowsException<T>(() => method()); | Assert.AreEqual(), Assert.IsTrue(), Assert.IsFalse() |
| **NUnit** | [TestFixture] (необязательно) | [Test] | [TestCase(...)] | [SetUp] | [TearDown] | Assert.Throws<T>(() => method()); | Assert.AreEqual(), Assert.IsTrue(), Assert.IsFalse() |
| **xUnit** | (Нет атрибута, класс может быть обычным) | [Fact] | [Theory] + [InlineData(...)] | **Конструктор класса** | **Деструктор класса** (Dispose()) | Assert.Throws<T>(() => method()); | Assert.Equal(), Assert.True(), Assert.False() |

## **Примеры использования команд**

| **Фреймворк** | **Пример простого теста** | **Пример параметризованного теста** |
| --- | --- | --- |
| **MSTest** | [**TestMethod**] public void Test() { Assert.AreEqual(4, 2 + 2); } | [**DataTestMethod**] [DataRow(2, 2, 4)] public void AddTest(int a, int b, int expected) { Assert.AreEqual(expected, a + b); } |
| **NUnit** | [**Test**] public void Test() { Assert.AreEqual(4, 2 + 2); } | [**TestCase**(2, 2, 4)] [TestCase(3, 3, 6)] public void AddTest(int a, int b, int expected) { Assert.AreEqual(expected, a + b); } |
| **xUnit** | [**Fact**] public void Test() { Assert.Equal(4, 2 + 2); } | [**Theory**] [InlineData(2, 2, 4)] [InlineData(3, 3, 6)] public void AddTest(int a, int b, int expected) { Assert.Equal(expected, a + b); } |