РОСЖЕЛДОР
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Ростовский государственный университет путей сообщения»
(ФГБОУ ВО РГУПС)
Тихорецкий техникум железнодорожного транспорта
(ТТЖТ – филиал РГУПС)

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

по организации самостоятельной работы обучающихся по МДК 02.03

«Разработка прикладных приложений»

для студентов специальности

09.02.01 Компьютерные системы и комплексы

III КУРС

Тихорецк, 2024 г.

УТВЕРЖДАЮ
Заместитель директора по УР
Н.Ю. Шитикова

Методические рекомендации по организации самостоятельной работы обучающимися по МДК 02.03 Разработка прикладных приложений для специальности 09.02.01 Компьютерные системы и комплексы.

Организация-разработчик: Тихорецкий техникум железнодорожного транспорта – филиал Федерального государственного бюджетного образовательного учреждения высшего образования «Ростовский государственный университет путей сообщения» (ТТЖТ – филиал РГУПС)

Разработчики:

Украинский А.В., преподаватель ТТЖТ - филиала РГУПС

Рекомендованы цикловой комиссией № 4 специальностей 09.02.01, 11.02.06, 38.02.01

Протокол заседания № 1 от «02» сентября 2024 г.

СОДЕРЖАНИЕ

1. ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Методические рекомендации для организации самостоятельной работы по МДК 02.03 Разработка прикладных приложений предназначены для обучающихся третьих курсов специальности 09.02.01 Компьютерные системы и комплексы.

В связи с введением в образовательный процесс нового Федерального государственного образовательного стандарта все более актуальной становится задача организации самостоятельной работы студентов. Самостоятельная работа определяется как индивидуальная или коллективная учебная деятельность, осуществляемая без непосредственного руководства преподавателя, но по его заданиям и под его контролем.

Самостоятельная работа студентов является одной из основных форм внеаудиторной работы при реализации учебных планов и программ. В материалах для самостоятельной работы студентов представлен курс поддержки и совершенствования общеобразовательных, коммуникативных, информационных компетенций, обеспечивающих практическое выполнение заданий (поиск, набор и обработка данных) и продуктивного плана.

Самостоятельная работа студентов проводится с целью:

- систематизации и закрепления полученных теоретических знаний и практических умений студентов;
- углубления и расширения теоретических знаний;
- развития познавательных способностей и активности студентов: самостоятельности, ответственности и организованности, творческой инициативы;
- формирования самостоятельности мышления, способности к саморазвитию, самосовершенствованию и самореализации.

В процессе выполнения самостоятельной работы студенты получают:

практические умения и навыки:

- умение оперировать данными на информационном рынке;

- умения работать с информацией (кодировать, представлять, измерять);
- умения обрабатывать информацию средствами информатики.

учебные умения:

- использовать различные информационные источники;
- расспрашивать, описывать, сравнивать, исследовать, анализировать оценивать;
- проводить самостоятельный поиск необходимой информации;

специальные учебные умения:

- осуществлять эффективный и быстрый поиск нужной информации;
- организовывать работу на компьютере;
- выбирать оптимальное программное обеспечение для работы с информацией;
- излагать информацию средствами информатики.

Самостоятельная работа может проходить в лекционном кабинете, во время внеклассных мероприятий, дома.

Количество часов, отведенное на самостоятельную работу, соответствует учебному плану.

2. ОСОБЕННОСТИ КУРСА И ОСНОВНЫЕ ТРЕБОВАНИЯ РАБОЧЕЙ ПРОГРАММЫ

1.1 Цель и планируемые результаты освоения профессионального модуля:

В результате изучения профессионального модуля, обучающихся должен освоить основной вид деятельности проектирование управляющих программ компьютерных систем и комплексов и соответствующие ему общие компетенции и профессиональные компетенции:

1.1.1. Перечень общих компетенций:

Код	Наименование общих компетенций
ОК 01	Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам
ОК 02	Использовать современные средства поиска, анализа и интерпретации информации, и информационные технологии для выполнения задач профессиональной деятельности.
ОК 03	Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях.
ОК 04	Эффективно взаимодействовать и работать в коллективе и команде.
ОК 05	Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста.
ОК 06	Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей, в том числе с учетом гармонизации межнациональных и межрелигиозных отношений, применять стандарты антикоррупционного поведения.
ОК 07	Содействовать сохранению окружающей среды, ресурсосбережению, применять знания об изменении климата, принципы бережливого производства, эффективно действовать в чрезвычайных ситуациях.
ОК 08	Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности.
ОК 09	Пользоваться профессиональной документацией на государственном и иностранном языках.

1.1.2. Перечень профессиональных компетенций:

Код	Наименование видов деятельности и профессиональных компетенций
ВД 2	Проектирование управляющих программ компьютерных систем и комплексов
ПК 2.1.	Проектировать, разрабатывать и отлаживать программный код модулей управляющих программ.
ПК 2.2.	Владеть методами командной разработки программных продуктов.
ПК 2.3.	Выполнять интеграцию модулей в управляющую программу.
ПК 2.4.	Тестировать и верифицировать выпуски управляющих программ.
ПК 2.5.	Выполнять установку и обновление версий управляющих программ (с учетом миграции - при необходимости).

3. ТРЕБОВАНИЯ К ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ И ПРАВИЛА ПОЛЬЗОВАНИЯ МЕТОДИЧЕСКИМИ РЕКОМЕНДАЦИЯМИ

Самостоятельная деятельность учащихся – внеаудиторная работа, предполагающая самостоятельное извлечение информации, её обработка (анализ и синтез), решение лингвистических задач.

Следовательно, требования к организации внеаудиторной деятельности касаются процесса поиска информации, источников информации и полученных на аудиторных занятиях предметных знаний, умений, навыков:

1. Умение пользоваться электронными ресурсами: сайтами, книгами и учебными пособиями, справочниками и словарями.
2. Умение пользоваться письменными источниками, находить нужную информацию.
3. Уметь работать с полученной информацией: обрабатывать её, сокращать, конспектировать.
4. Учащиеся должны вести записи в лекционной тетради и уместно использовать записи при решении заданий самостоятельной работы.
5. Учащиеся должны выполнять требования к каждому заданию данных методических рекомендаций.

Структура методических рекомендаций создана таким образом, чтобы максимально облегчить работу студентам и преподавателям. Пользоваться данными указаниями несложно: необходимо внимательно прочесть требования к выполнению заданий, просмотреть списки рекомендуемой литературы.

4. ВИДЫ САМОСТОЯТЕЛЬНЫХ РАБОТ И ФОРМЫ КОНТРОЛЯ

В учебном процессе выделяют два вида самостоятельной работы:

- аудиторная;
- внеаудиторная.

Аудиторная самостоятельная работа по дисциплине выполняется на учебных занятиях под непосредственным руководством преподавателя и по его заданию.

Внеаудиторная самостоятельная работа выполняется студентом по заданию преподавателя, но без его непосредственного участия.

Содержание внеаудиторной самостоятельной определяется в соответствии с рекомендуемыми видами заданий согласно примерной и рабочей программ учебной дисциплины.

Виды заданий для самостоятельной работы:

На основании компетентностного подхода к реализации профессиональных образовательных программ, видами заданий для внеаудиторной самостоятельной работы являются:

Для овладения знаниями: поиск информации в сети Интернета, проведение исследований, подготовка сообщений.

Для закрепления и систематизации знаний: применение различных специализированных программ.

Для формирования умений: обработка информации прикладными программами, проектирование и моделирование объектов.

Формы самостоятельной работы:

Поиск информации в различных источниках и ее практическая обработка.

Самостоятельная работа в виде выполнения заданий.

Составление информационных моделей объектов и их анализ.

Критерии оценки результатов внеаудиторной самостоятельной работы студентов:

- уровень освоения студентами учебного материала;
- умение студента использовать теоретические знания при выполнении практических задач;
- сформированность общеучебных умений;
- обоснованность и четкость изложения ответа;
- оформление материала в соответствии с требованиями.

Контроль выполненной самостоятельной работы осуществляется индивидуально, на уроке, при тестировании, на семинаре, при защите рефератов и проектов:

Контроль сообщений осуществляется на уроках.

Контроль выполнения рефератов осуществляется индивидуальной (или групповой) беседой по ключевым моментам работы, с последующей защитой реферата.

Проверка самостоятельных работ информационных моделей объектов проверяется индивидуально. Самостоятельная работа может осуществляться индивидуально или группами студентов в зависимости от цели, объема, конкретной тематики самостоятельной работы, уровня сложности, уровня умений студентов.

Контроль результатов внеаудиторной самостоятельной работы студентов может осуществляться в пределах времени, отведенного на обязательные учебные занятия по дисциплине и внеаудиторную самостоятельную работу студентов по дисциплине, может проходить в письменной, устной или смешанной форме.

5. ХАРАКТЕРИСТИКА ЗАДАНИЯ

1. Написание реферата – это более объемный, чем сообщение, вид самостоятельной работы студента, содержащий информацию, дополняющую и развивающую основную тему, изучаемую на аудиторных занятиях *(приложение*). Ведущее место занимают темы, представляющие профессиональный интерес, несущие элемент новизны. Реферативные материалы должны представлять письменную модель первичного документа – научной работы, монографии, статьи. Реферат может включать обзор нескольких источников и служить основой для доклада на определенную тему на семинарах, конференциях.

Регламент озвучивания реферата – 7-10 мин.

Затраты времени на подготовку материала зависят от трудности сбора информации, сложности материала по теме, индивидуальных особенностей студента и определяются преподавателем. Ориентировочное время на подготовку – 4 ч.

Порядок сдачи и защиты рефератов.

1. Реферат сдается на проверку преподавателю за 1-2 недели до зачетного занятия

2. При оценке реферата преподаватель учитывает

- качество
- степень самостоятельности студента и проявленную инициативу
- связность, логичность и грамотность составления
- оформление в соответствии с требованиями ГОСТ.

3. Защита тематического реферата может проводиться на выделенном одном занятии в рамках часов учебной дисциплины или конференции или по одному реферату при изучении соответствующей темы, либо по договоренности с преподавателем.

4. Защита реферата студентом предусматривает

- доклад по реферату не более 5-7 минут
- ответы на вопросы оппонента.

На защите запрещено чтение текста реферата.

Общая оценка за реферат выставляется с учетом оценок за работу, доклад, умение вести дискуссию и ответы на вопросы. Содержание и оформление разделов реферата.

Титульный лист является первой страницей реферата и заполняется по строго определенным правилам.

В верхнем поле указывается полное наименование учебного заведения.

В среднем поле дается заглавие реферата, которое проводится без слова «тема» и в кавычки не заключается.

Далее, ближе к правому краю титульного листа, указываются фамилия, инициалы студента, написавшего реферат, а также его курс и группа. Немного ниже или слева указываются название кафедры, фамилия и инициалы преподавателя - руководителя работы.

В нижнем поле указывается год написания реферата.

После титульного листа помещают оглавление, в котором приводятся все заголовки работы и указываются страницы, с которых они начинаются. Заголовки оглавления должны точно повторять заголовки в тексте. Сокращать их или давать в другой формулировке и последовательности нельзя.

Все заголовки начинаются с прописной буквы без точки на конце. Последнее слово каждого заголовка соединяют отточием (……) с соответствующим ему номером страницы в правом столбце оглавления.

Заголовки одинаковых ступеней рубрикации необходимо располагать друг под другом. Заголовки каждой последующей ступени смещают на три - пять знаков вправо по отношению к заголовкам предыдущей ступени.

Введение. Здесь обычно обосновывается актуальность выбранной темы, цель и содержание реферата, указывается объект (предмет) рассмотрения, приводится характеристика источников для написания работы и краткий обзор имеющейся по данной теме литературы. Актуальность предполагает оценку своевременности и социальной значимости выбранной темы, обзор литературы по теме отражает знакомство автора реферата с имеющимися источниками, умение их систематизировать, критически рассматривать, выделять существенное, определять главное.

Основная часть. Содержание глав этой части должно точно соответствовать теме работы и полностью ее раскрывать. Эти главы должны показать умение исследователя сжато, логично и аргументировано излагать материал, обобщать, анализировать, делать логические выводы.

Заключительная часть. Предполагает последовательное, логически стройное изложение обобщенных выводов по рассматриваемой теме.

Библиографический список использованной литературы составляет одну из частей работы, отражающей самостоятельную творческую работу автора, позволяет судить о степени фундаментальности данного реферата.

В работах используются следующие способы построения библиографических списков: по алфавиту фамилий, авторов или заглавий; по тематике; по видам изданий; по характеру содержания; списки смешанного построения. Литература в списке указывается в алфавитном порядке (более распространенный вариант - фамилии авторов в алфавитном порядке), после указания фамилии и инициалов автора указывается название литературного источника, место издания (пишется сокращенно, например, Москва - М., Санкт - Петербург - СПб ит.д.), название издательства (например, Мир), год издания (например, 2022), можно указать страницы (например, с. 54-67). Страницы можно указывать прямо в тексте, после указания номера, под которым литературный источник находится в списке литературы (например, 7 (номер лит. источника), с. 67- 89). Номер литературного источника указывается после каждого нового отрывка текста из другого литературного источника.

В приложении помещают вспомогательные или дополнительные материалы, которые загромождают текст основной части работы (таблицы, карты, графики, неопубликованные документы, переписка и т.д.). Каждое приложение должно начинаться с нового листа (страницы) с указанием в правом верхнем углу слова " Приложение" и иметь тематический заголовок. При наличии в работе более одного приложения они нумеруются арабскими цифрами (без знака " № "), например, " Приложение 1". Нумерация страниц, на которых даются приложения, должна быть сквозной и продолжать общую нумерацию страниц основного текста. Связь основного текста с приложениями осуществляется через ссылки,

которые употребляются со словом " смотри " (оно обычно сокращается и заключается вместе с шифром в круглые скобки - (см. прил. 1)).

Критерии оценки реферата

- актуальность темы, 1 балл;
- соответствие содержания теме, 3 балла;
- глубина проработки материала, 3 балла;
- грамотность и полнота использования источников, 1 балл;
- соответствие оформления реферата требованиям, 2 балла;
- доклад, 5 баллов;
- умение вести дискуссию и ответы на вопросы, 5 баллов.

Максимальное количество баллов: 20.
19-20 баллов соответствует оценке «5»
15-18 баллов – «4»
10-14 баллов – «3»
менее 10 баллов – «2»

2. Создание материалов-презентаций – это вид самостоятельной работы студентов по созданию наглядных информационных пособий, выполненных с помощью мультимедийной компьютерной программы. Этот вид работы требует координации навыков студента по сбору, систематизации, переработке информации, оформления ее в виде подборки материалов, кратко отражающих основные вопросы изучаемой темы, в электронном виде. То есть создание материалов-презентаций расширяет методы и средства обработки и представления учебной информации, формирует у студентов навыки работы на компьютере.

Материалы-презентации готовятся студентом в виде слайдов с использованием программы для презентаций. В качестве материалов-презентаций могут быть представлены результаты любого вида внеаудиторной самостоятельной работы, по формату соответствующие режиму презентаций.

Затраты времени на создание презентаций зависят от степени трудности материала по теме, его объема, уровня сложности создания презентации, индивидуальных особенностей студента и определяются преподавателем.

Ориентировочное время на подготовку – 1 ч.

Критерии оценки

- соответствие содержания теме, 1 балл;
- правильная структурированность информации, 5 баллов;
- наличие логической связи изложенной информации, 5 балл;
- эстетичность оформления, его соответствие требованиям, 3 балла;
- работа представлена в срок, 1 балл.

Максимальное количество баллов: 15.
14-15 баллов соответствует оценке «5»
11-13 баллов – «4»
8-10 баллов – «3»
менее 8 баллов – «2»

3. Подготовка презентация и доклада.

Доклад – это сообщение по заданной теме, с целью внести знания из дополнительной литературы, систематизировать материл, проиллюстрировать примерами, развивать навыки самостоятельной работы с научной литературой, познавательный интерес к научному познанию.

Тема доклада должна быть согласована с преподавателем и соответствовать теме занятия.

Материалы при его подготовке, должны соответствовать научно-методическим требованиям учебного заведения и быть указаны в докладе.

Необходимо соблюдать регламент, оговоренный при получении задания.

Иллюстрации должны быть достаточными, но не чрезмерными.

Работа студента над докладом-презентацией включает отработку навыков ораторства и умения организовать и проводить диспут.

Студент в ходе работы по презентации доклада, отрабатывает умение ориентироваться в материале и отвечать на дополнительные вопросы слушателей.

Студент в ходе работы по презентации доклада, отрабатывает умение

самостоятельно обобщить материал и сделать выводы в заключении.

Докладом также может стать презентация реферата студента, соответствующая теме занятия.

Студент обязан подготовить и выступить с докладом в строго отведенное время преподавателем, и в срок.

Необходимо помнить, что выступление состоит из трех частей: вступление, основная часть и заключение.

Вступление помогает обеспечить успех выступления по любой тематике. Вступление должно содержать:

- название презентации (доклада)
- сообщение основной идеи
- современную оценку предмета изложения
- краткое перечисление рассматриваемых вопросов
- живую интересную форму изложения
- акцентирование оригинальности подхода

Основная часть, в которой выступающий должен глубоко раскрыть суть затронутой темы, обычно строится по принципу отчета. Задача основной части - представить достаточно данных для того, чтобы слушатели и заинтересовались темой и захотели ознакомиться с материалами. При этом логическая структура теоретического блока не должна даваться без наглядных пособий, аудио-визуальных и визуальных материалов.

Заключение - это ясное четкое обобщение и краткие выводы, которых всегда ждут слушатели.

Примерный план публичного выступления

1. Приветствие

«Добрый день!»

«Уважаемый «(имя и отчество преподавателя)

«Уважаемые присутствующие!»

2. Представление (Ф.И., группа, и т.д.)

«Меня зовут...Я учащийся (-щаяся).группы, техникума №..., города....»

3. Цель выступления

«Цель моего выступления – дать новую информацию по теме.

4. Название темы

«Название темы»

5.Актуальность

«Актуальность и выбор темы определены следующими факторами: во-первых,..., во-вторых,...»

6. Кратко о поставленной цели и способах ее достижения

«Цель моего выступления – ... основные задачи и способы их решения: 1..., 2..., 3...»

получены новые знания следующего характера:...,

выдвинуты новые гипотезы и идеи:...,

определены новые проблемы (задачи)»

7. Благодарность за внимание

«Благодарю за проявленное внимание к моему выступлению»

8. Ответы на вопросы

«Спасибо (благодарю) за вопрос...

А) Мой ответ...

Б) У меня, к сожалению, нет ответа, т.к. рассмотрение данного вопроса не входило в задачи моего исследования.

9. Благодарность за интерес и вопросы по теме

«Благодарю за интерес и вопросы по подготовленной теме. Всего доброго»

Факторы, влияющие на успех выступления

До, во время и после выступления на конференции докладчику необходимо учесть существенные факторы, непосредственно связанные с формой выступления - это внешний вид и речь докладчика, используемый демонстрационный материал, а также формы ответов на вопросы в ходе выступления.

Речь

Громкость – доступная для восприятия слов отдаленными слушателями, но без крика и надрыва.

Произношение слов – внятное, четкое, уверенное, полное (без глотания окончаний), с правильным литературным ударением.

Темп – медленный – в значимых зонах информации, средний – в основном изложении, быстрый – во вспомогательной информации.

Интонация – дружественная, спокойная, убедительная, выразительная, без ироничных и оскорбительных оттенков.

Критерии оценки доклада

- актуальность темы, 1 балл;
- соответствие содержания теме, 1 балл;
- глубина проработки материала, 1 балл;
- грамотность и полнота использования источников, 1 балл;
- соответствие оформления доклада требованиям, 1 балл.
- умение вести дискуссию и ответы на вопросы, 5 баллов.

Максимальное количество баллов: 10.

9-10 баллов соответствует оценке «5»

7-8 баллов – «4»

5-7 баллов – «3»

менее 5 баллов – «2»

4. Решение задач

Решение задачи можно условно разбить на четыре этапа и в соответствии с данными этапами установить критерии оценки:

1. Ознакомиться с условием задачи (анализ условия задачи и его наглядная интерпретация схемой или чертежом), 0,5 балл.
2. Составить план решения задачи (составление уравнений, связывающих физические величины, которые характеризуют рассматриваемое явление с количественной стороны), 2 балла;
3. Осуществить решение (совместное решение полученных уравнений относительно той или иной величины, считающейся в данной задаче неизвестной), 2 балла;
4. Проверка правильности решения задачи (анализ полученного результата и числовой расчет), 0,5 балла.

Максимальное количество баллов: 5.

Оценка выставляется по количеству набранных баллов. - Проведение контрольных и проверочных работ, включающих в себя задания самостоятельных;

- устный опрос;
- проверка выполненного материала

5.Выполнение тестовых заданий

Тесты и задания сориентированы на проверку выполнения обязательных требований к уровню общеобразовательной подготовки по физике.

Система заданий возрастающей степени трудности и специфической формы позволяет качественно оценить структуру и определить уровень знаний.

Тест состоит из следующих частей:

1. Одинаковая инструкция для всех испытуемых, которая должна быть настолько проста и понятна, насколько это возможно.

2. Сами тестовые задания: а) задания открытой формы. Инструкцией к заданиям данного типа является одно слово «дополните». За правильный ответ студент получает один балл. б) задания закрытой формы – вопрос с вариантами ответов, один или несколько из которых правильные. Неправильные ответы должны быть такими, чтобы каждый из них мог привлечь внимание. Инструкцией к этому типу заданий является: «выберите один (несколько) правильных ответов». За правильный ответ студент получает один балл. в) задания на восстановление соответствия. Инструкцией является: «установите соответствие». Число баллов оценивается отдельно, причём число баллов равно числу правильно установленных соответствий. Студент, допустивший хотя бы одну ошибку, получает 0 баллов. г) задания на установление правильной последовательности. Инструкцией является: «установите правильную последовательность». Если ранги в расставлены правильно – студент получает один балл, если допущена хотя бы одна ошибка – ноль баллов.

3. Одинаковые правила оценки ответов в рамках принятой формы. Оценка «удовлетворительно» ставится, если студент набирает не менее 50% баллов и до 75%. Оценка «хорошо» - 76 – 90% заданий. Оценка «отлично» - 90% и выше.

Структура теста удовлетворяет следующим основным требованиям:

- задания каждого типа располагаются в одном месте и в порядке возрастания сложности;
- задания формулируются в логической форме высказывания;
- основной текст задания содержит не более 7 – 8 слов;
- задания должны выполняться быстро, не более, чем за 1-2 минуты.

При организации тематического контроля знаний необходимо учитывать, что:

- тематическая тестовая контрольная работа обычно рассчитана на 30-45 минут;
- имеет 4 варианта;
- каждый вариант имеет 15 заданий, правильные ответы на которые предполагают усвоение учебного материала, содержание заданий должно включать все основные понятия, законы и явления, необходимые для усвоения.

6. ТЕМАТИЧЕСКИЙ ПЛАН САМОСТОЯТЕЛЬНОЙ РАБОТЫ

№ п.п	Тема	Объём часов с.р.	Задание	Деятельность студента	Форма контроля
1	Интернет вещей (IoT)	2 часа	Ср № 1	Поисково-исполнительский и творческий	Отчет о проведении работы
2	Среды разработки для мобильных платформ и ПК	2 часа	Ср № 2	Репродуктивно-подражающий	Устный опрос, проверка материала
3	Языки программирования для разработки приложений	2 часа	Ср № 3	Активизация познавательной деятельности	Устный опрос, проверка материала
4	Особенности языка программирования Java	2 часа	Ср № 4	Репродуктивно-подражающий	Устный опрос, проверка материала
5	Особенности языка программирования C	2 часа	Ср № 5	Поисково-исполнительский и творческий	Отчет о проведении работы
6	Оператор switch. Циклы	2 часа	Ср № 6	Репродуктивно-подражающий	Устный опрос, проверка материала
7	Массивы в языке C	2 часа	Ср № 7	Активизация познавательной деятельности	Устный опрос, проверка материала
8	Математические вычисления в языке C	2 часа	Ср № 8	Репродуктивно-подражающий	Устный опрос, проверка материала
9	Символы и строки в языке C	2 часа	Ср № 9	Поисково-исполнительский и творческий	Отчет о проведении работы
10	Поиск, сравнение при обработке символов и строк	2 часа	Ср № 10	Репродуктивно-подражающий	Устный опрос, проверка материала
11	Обзор основных принципов ООП	2 часа	Ср № 11	Активизация познавательной деятельности	Устный опрос, проверка материала
12	Понятие класса	2 часа	Ср № 12	Репродуктивно-подражающий	Устный опрос, проверка материала
13	Модификаторы	2 часа	Ср № 13	Репродуктивно-подражающий	Устный опрос, проверка материала
14	Использование пакетов, директив импорта	2 часа	Ср № 14	Поисково-исполнительский и творческий	Отчет о проведении работы
15	Расширение и инкапсуляция свойств класса	2 часа	Ср № 15	Репродуктивно-подражающий	Устный опрос, проверка материала

16	Наследование как механизм повторного использования кода	2 часа	Ср № 16	Активизация познавательной деятельности	Устный опрос, проверка материала
17	Конструктор при наследовании свойств и методов класса	2 часа	Ср № 17	Поисково-исполнительский и творческий	Отчет о проведении работы
18	Виртуальные методы и позднее связывание	4 часа	Ср № 18	Поисково-исполнительский и творческий	Отчет о проведении работы

7. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Самостоятельная работа № 1 (2 часа)

Тема: Интернет вещей (IoT).
Цель работы: Понять концепцию Интернета вещей, практические аспекты его реализации.

Краткие сведения из теории

Интернет вещей (IoT) — концепция, согласно которой различные устройства и предметы могут соединяться друг с другом и с интернетом, что позволяет им собирать, обмениваться данными и взаимодействовать с окружающей средой. Приложения IoT охватывают различные сферы, такие как:

1. Умные устройства для дома: термостаты, освещение, системы безопасности.
2. Промышленный IoT: мониторинг производственных процессов, предсказательное обслуживание оборудования.
3. Здравоохранение: носимые устройства для контроля состояния здоровья, удаленный мониторинг пациентов.
4. Умные города: управление ресурсами (водоснабжение, освещение), транспортные системы.

Средства разработки приложений IoT
Для разработки приложений IoT используются различные платформы и инструменты:

1. Аппаратные средства:
    - Arduino, Raspberry Pi, ESP8266 и другие микроконтроллеры и одноплатные компьютеры.
2. Программные средства:
    - Языки программирования: Python, C/C++, Java, JavaScript.
    - Платформы: Node-RED, ThingSpeak, OpenHAB для визуализации данных и управления устройствами.
3. Облачные решения:
    - Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform предлагают инструменты для хранения и обработки больших данных, получаемых от устройств.
4. Протоколы связи:
    - MQTT, CoAP, HTTP/HTTPS для передачи данных между устр-ми и серверами.

Задания для выполнения студентами

1. Проектирование простого IoT устройства:
    - Выберите одну из сфер применения IoT и спроектируйте устройство для решения конкретной задачи. Опишите аппаратные и программные средства, которые будете использовать.
2. Разработка простого приложения:
    - Создайте приложение на платформе Arduino, которое будет считывать данные с датчика (например, температуры) и передавать их по протоколу MQTT на облачный сервер.
3. Анализ существующих приложений:
    - Исследуйте три существующих IoT-приложения. Определите их преимущества и недостатки, а также технологическую инфраструктуру.

Контрольные вопросы

1. Что такое Интернет вещей и каковы его ключевые характеристики?
2. Какие основные компоненты включает в себя типичная архитектура IoT-решения?
3. Каковы преимущества использования облачных решений в приложениях IoT?
4. Какие протоколы связи чаще всего используются в IoT и в чем их особенности?
5. Приведите примеры применения IoT в регионе и их влияние на повседневную жизнь.

Самостоятельная работа № 2 (2 часа)

Тема: Среды разработки для мобильных платформ и ПК.
Цель работы: Изучить инструменты для создания приложений на различных платформах.

Краткие сведения из теории

Среды разработки (Integrated Development Environments, IDE) обеспечивают разработчикам удобный интерфейс для разработки, отладки и тестирования программного обеспечения. Использование специализированных сред значительно упрощает процесс разработки и помогает избежать ошибок. Существует множество сред разработки, адаптированных для различных платформ и языков программирования.

Основные типы сред разработки

1. Для мобильных платформ:
    - Android Studio: Официальная среда разработки для приложений на Android. Поддерживает Java, Kotlin и различные Android API.
    - Xcode: Основная среда для разработки приложений под iOS и macOS. Поддерживает Swift и Objective-C.
    - Flutter: Платформенно-Agnostic фреймворк для разработки кроссплатформенных приложений (iOS, Android, Web) с использованием языка Dart.
    - React Native: Популярная библиотека для создания мобильных приложений на JavaScript, позволяющая использовать компоненты React.
2. Для ПК:
    - Visual Studio: Мощная IDE от Microsoft, поддерживающая различные языки (C#, C++, VB.NET) и предназначенная для разработки под Windows.
    - IntelliJ IDEA: Профессиональная IDE для разработки на Java и других языках, предлагающая расширенные функции для работы с большими проектами.
    - Eclipse: Открытое решение для разработки на Java и других языках. Часто используется в образовательных учреждениях.
    - PyCharm: Специально разработанная для Python IDE, обеспечивающая удобство разработки и отладки.

Задания для выполнения студентами

1. Сравнительный анализ IDE:
    - Выберите две среды разработки (одну для мобильных платформ и одну для ПК). Проведите сравнительный анализ их возможностей и функционала, описывая преимущества и недостатки каждой из них.
2. Создание простого приложения:
    - Используя одну из выбранных IDE (например, Android Studio или Visual Studio), разработайте простое приложение/проект. Для мобильной платформы это может быть "Hello World" приложение, для ПК — консольное приложение.
3. Изучение инструментов отладки:
    - Воспользуйтесь инструментами отладки в вашей среде разработки. Создайте небольшую программу с ошибками и изучите, как использовать дебаггер для их поиска и устранения.

Контрольные вопросы

1. Какие основные функции должна обеспечивать среда разработки?
2. В чем отличие между нативными и кроссплатформенными средами разработки?
3. Назовите особенности Android Studio и Xcode, которые делают их подходящими для разработки приложений на соответствующих платформах.
4. Какие факторы следует учитывать при выборе среды разработки для конкретного проекта?
5. Какие возможности отладки предоставляет Visual Studio и какие инструменты могут помочь разработчику в этом процессе?

Самостоятельная работа № 3 (2 часа)

Тема: Языки программирования для разработки приложений.
Цель работы: Изучить возможность использования языков программирования в разных областях.

Краткие сведения из теории

Языки программирования являются основой разработки приложений. Каждый язык имеет свои особенности, преимущества и недостатки, а также применение в различных областях. Рассмотрим четыре популярных языка программирования: C++, C#, Java и Python.

C++

- Особенности: C++ — это компилируемый язык, который поддерживает объектно-ориентированное, процедурное и обобщенное программирование. Проектируется для высокопроизводительных приложений.
- Применимость: Shirokiy спектр применения, включая системное программирование, разработку игр, драйвера, а также приложения с высокими требованиями к производительности.
- Достоинства: Высокая производительность, доступ к низкоуровневым ресурсам, богатая стандартная библиотека.
- Недостатки: Сложность освоения, легче допустить ошибки (например, в управлении памятью), требует более детального управления ресурсами.

C#

- Особенности: C# — это язык, разработанный Microsoft как часть платформы .NET. Поддерживает объектно-ориентированное программирование.
- Применимость: Широко используется для разработки приложений под Windows, веб-приложений (ASP.NET), а также игр (Unity).
- Достоинства: Простой синтаксис, хорошее взаимодействие с .NET Framework, мощные инструменты разработки (например, Visual Studio).
- Недостатки: Ограниченная кроссплатформенность (хотя .NET Core улучшает ситуацию), зависимость от экосистемы Microsoft.

Java

- Особенности: Java — это высокоуровневый объектно-ориентированный язык программирования, который работает на принципе "один раз написано — везде запущено" благодаря JVM (Java Virtual Machine).
- Применимость: Используется для серверной разработки, веб-приложений, мобильных приложений (Android) и больших систем (например, корпоративные решения).
- Достоинства: Портативность, поддержка многопоточности, обширная экосистема библиотек и фреймворков.
- Недостатки: Меньшая производительность по сравнению с C++, необходимость в виртуальной машине, которая потребляет дополнительные ресурсы.

Python

- Особенности: Python — интерпретируемый язык, известный своим простым и читаемым синтаксисом, поддерживающий множество парадигм программирования.
- Применимость: Широко используется в веб-разработке, научных вычислениях, автоматизации, разработке игр и анализе данных.
- Достоинства: Удобство в изучении, большое сообщество, множество библиотек и фреймворков (например, Django, Flask, Pandas).
- Недостатки: Меньшая скорость выполнения по сравнению с компилируемыми языками, ограниченная производительность для системного программирования и приложений с высокими требованиями.

Задания для выполнения студентами

1. Исследование языков: Выберите один из языков программирования (C++, C#, Java, Python) и подготовьте презентацию о его особенностях, применимости, достоинствах и недостатках. Укажите примеры популярных приложений, разработанных на этом языке.
2. Сравнительный анализ: Сравните два выбранных языка программирования по трём ключевым критериям (например, производительность, легкость обучения, область применения). Подготовьте отчет.
3. Практическое задание: Напишите простую программу на выбранном языке, которая выполняет какую-либо задачу (например, калькулятор или программу для работы с массивами). Прокомментируйте код и объясните, почему выбрали именно этот язык.

Контрольные вопросы

1. Каковы основные отличия между компилируемыми и интерпретируемыми языками программирования?
2. Для каких типов приложений лучше всего подходит каждый из представленных языков программирования?
3. Какие факторы могут повлиять на выбор языка программирования для конкретного проекта?
4. Каковы основные причины популярности Python среди начинающих программистов?
5. В чем заключаются преимущества и недостатки использования Java в разработке корпоративных приложений?

Самостоятельная работа № 4 (2 часа)

Тема: Особенности языка программирования Java.
Цель работы: Познакомиться с основами Java-технологий, изучить особенности языка и усвоить навыки использования интегрированных сред разработки.

Краткие сведения из теории

Java — это объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (сейчас часть Oracle) в середине 1990-х годов. Он был создан с целью обеспечить высокую производительность, портативность и безопасность. Основная идея Java заключается в принципе "один раз написано — везде будет работать", что достигается благодаря использованию Java Virtual Machine (JVM).
Особенности языка программирования Java

1. Портативность: Java-программы могут выполняться на любой платформе, где установлена JVM. Это позволяет разработчикам создавать универсальные приложения.
2. Объектно-ориентированность: Java поддерживает основные принципы ООП — инкапсуляцию, наследование и полиморфизм.
3. Безопасность: Java обеспечивает безопасность на нескольких уровнях, включая возможность ограничения доступа к данным и защиты от выполнения вредоносных программ.
4. Управление памятью: Java использует автоматический механизм сборки мусора, что облегчает управление памятью и снижает вероятность утечек памяти.
5. Многопоточность: Java предлагает встроенные средства для работы с потоками, что упрощает разработку многозадачных приложений.

Описание Java технологий
Java предоставляет множество технологий, способствующих разработке различных типов приложений, включая:

- Java SE (Standard Edition): Базовая платформа для разработки настольных приложений и утилит.
- Java EE (Enterprise Edition): Платформа для создания корпоративных приложений, использующая технологии сервлетов, JSP, EJB, и многое другое.
- Java ME (Micro Edition): Разработка приложений для мобильных устройств и встроенных систем.
- JavaFX: Технология для создания современных клиентских приложений с графическим интерфейсом.

Использование интегрированной среды разработки (IDE)
Использование интегрированных сред разработки значительно упрощает процесс программирования на Java. Некоторые из популярных IDE для Java:

- Eclipse: Мощная и расширяемая среда с открытым исходным кодом, поддерживающая множество плагинов.
- IntelliJ IDEA: Профессиональная IDE с мощными возможностями рефакторинга и интеллектуальной поддержкой кода.
- NetBeans: Простой в использовании IDE, который подходит для начинающих и поддерживает все Java технологии.

Задания для выполнения студентами

1. Изучение основ Java:
    - Напишите небольшой отчет о принципах объектно-ориентированного программирования в Java. Приведите примеры использования классов и объектов.
2. Технологии Java:

- Составьте таблицу, в которой перечислены основные технологии Java (SE, EE, ME, FX) с описанием их особенностей и областей применения.

3. Практическое задание:
    - Выберите одну из интегрированных сред разработки (Eclipse, IntelliJ IDEA или NetBeans). Разработайте простое приложение "Hello World" на Java. Опишите процесс установки IDE и создания проекта.
4. Пример кода:
    - Напишите код небольшого Java-приложения, использующего классы и методы. Описывайте, как вы реализовали основные функции.

Контрольные вопросы

1. Каковы основные преимущества использования языка программирования Java по сравнению с другими языками?
2. Что такое JVM и почему она важна для программирования на Java?
3. Назовите основные различия между Java SE, EE и ME.
4. Каковы главные принципы объектно-ориентированного программирования, и как они реализованы в Java?
5. Какие функции и возможности предлагают современные IDE для разработки на Java?

Самостоятельная работа № 5 (2 часа)

Тема: Особенности языка программирования С.
Цель работы: Углубить понимание языка С, его особенности и технологии, обеспечивающие эффективную разработку приложений.

Краткие сведения из теории

Язык программирования С был разработан в начале 1970-х годов для создания операционных систем и приложений, требующих высокой производительности. Он стал основой для создания многих других языков, таких как C++, C#, и Java.
Особенности языка программирования С

1. Производительность:
    - С позволяет программам работать быстро за счет низкоуровневого доступа к памяти и аппаратным ресурсам. Это делает язык идеальным для системного программирования.
2. Портативность:
    - Программы на С можно легко переносить на различные платформы. Стандарт ANSI C обеспечивает совместимость между различными компиляторами.
3. Строгая типизация:
    - С требует явного указания типов данных, что снижает вероятность ошибок в коде, особенно при работе с указателями.
4. Минимализм:
    - Язык имеет простой и лаконичный синтаксис, что позволяет разрабатывать компактные и эффективные программы.
5. Поддержка структурированного программирования:
    - С поддерживает функции, массивы, структуры и указатели, что позволяет создавать сложные программы с организованной логикой.

Описание С технологий

- Стандарты языка:
    - ANSI C: Общепринятый стандарт, задающий основы синтаксиса и семантики языка.
    - C99: Обновленный стандарт с новыми функциями, такими как переменные в блоках.
    - C11 и последующие: Включает усовершенствования для многопоточности и работы с памятью.
- Кросс-платформенная поддержка:
    - С технологии обеспечивают разработку приложений, которые могут работать на различных вычислительных платформах.

Использование интегрированной среды разработки (IDE)
Интегрированные среды разработки значительно упрощают процесс программирования. Популярные IDE для языка С:

- Code::Blocks: Простая и расширяемая среда с поддержкой множества компиляторов.
- Dev-C++: Легкая IDE, идеальная для начинающих.
- Eclipse CDT: Мощная платформа для разработки с поддержкой множества инструментов.
- CLion: Профессиональная среда от JetBrains с современными функциями.

Задания для выполнения студентами

1. Изучение особенностей С:
    - Напишите краткий отчет, освещающий 3-4 основных особенности языка С. Приведите примеры их применения.
2. Стандарты С:

- Создайте таблицу, в которой перечислены основные версии стандарта C и их ключевые особенности (ANSI C, C99, C11 и последующие).

3. Установка IDE:
   - Установите одну из IDE (например, Code::Blocks или Dev-C++) и создайте простую программу на C, выводящую "Hello, World!". Запишите процесс установки и настройки среды.
4. Пример кода:
   - Напишите простую программу на C, которая запрашивает у пользователя ввод числа и выводит его квадрат. Прокомментируйте ваш код.
5. Использование стандартных библиотек:
   - Создайте программу, использующую функцию из библиотеки math.h для вычисления квадратного корня и выводящую результат на экран.

Контрольные вопросы

1. Какие преимущества языка C делают его идеальным для системного программирования?
2. В чем состоят главные отличия стандартов ANSI C и C99?
3. Какова роль интегрированных сред разработки в процессе программирования на C?
4. Объясните, что такое указатели в языке C и как они используются.
5. Почему важно следить за стандартами языка при разработке программного обеспечения?

Самостоятельная работа № 6 (2 часа)

Тема: Оператор switch. Циклы.
Цель работы: Освоить синтаксис и применение различных операторов и циклов, что является основой программирования и логики в приложениях.

Краткие сведения из теории

Оператор switch — это конструкция для выбора одного из множества значений на основе выражения. Он часто используется как альтернатива множественным if-операторам. Синтаксис позволяет упрощать код, вместо того чтобы использовать несколько условных операторов.
Пример синтаксиса:

```
switch (expression) {
   case value1:
      // блок кода
      break;
   case value2:
      // блок кода
      break;
   default:
      // блок кода
}
```

Цикл for
Цикл for используется для повторного выполнения блока кода заданное количество раз. Он включает в себя инициализацию, условие продолжения и итерацию.
Пример синтаксиса:
c

```
for (initialization; condition; iteration) {
   // блок кода
}
Пример использования:
c
Copy
for (int i = 0; i < 10; i++) {
   printf("%d\n", i);
}
```

Бесконечный цикл
Бесконечный цикл — это цикл, который никогда не завершает выполнение, если только не происходит прерывание программы. Он может быть реализован с помощью for, while или do...while.
Пример:

```
while (true) {
   // блок кода
}
```

Цикл foreach
foreach — это конструкция для перебора коллекций. В некоторых языках (например, C# и Java) foreach позволяет легко проходить по элементам массива или коллекции.
Пример в C#:

```
foreach (var item in collection) {
   // операции с item
```

```
}
```

Вложенные циклы

Вложенные циклы позволяют разместить один цикл внутри другого, что может быть полезно для работы с многомерными массивами или сложными структурами данных.

Пример:

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        printf("%d %d\n", i, j);
    }
}
```

Цикл while

Цикл while используется для выполнения блока кода, пока условие истинно. Обычно он применим там, где число итераций заранее неизвестно.

Пример синтаксиса:

```
while (condition) {
    // блок кода
}
```

Задания для выполнения студентами

1. Изучение операторов и циклов:
    - Напишите краткий отчет о каждом из операторов и циклов, описанных выше. Приведите примеры из практики.
2. Программа с использованием switch:
    - Напишите программу, которая запрашивает у пользователя число от 1 до 5 и выводит его текстовое представление с помощью оператора switch.
3. Цикл for:
    - Создайте программу, которая выводит на экран первые 10 чисел Фибоначчи, используя цикл for.
4. Бесконечный цикл:
    - Разработайте программу, которая принимает ввод пользователя в бесконечном цикле до тех пор, пока пользователь не введет слово "выход".
5. Цикл foreach:
    - Если возможно, используйте foreach для перебора массива строк и вывода каждой строки на экран (если вы используете C#, используйте язык с поддержкой foreach).
6. Вложенные циклы:
    - Напишите программу, которая создает двумерный массив и заполняет его случайными числами. Затем, используя вложенные циклы, выведите элементы массива на экран.
7. Цикл while:
    - Напишите программу, которая запрашивает у пользователя ввод чисел и вычисляет их сумму. Цикл должен продолжаться до тех пор, пока пользователь не введет число 0.

Контрольные вопросы

1. В чем отличие оператора switch от множества if-операторов?
2. Когда целесообразно использовать бесконечный цикл, и какую опасность он может представлять?
3. Объясните разницу между циклами for, while и foreach.
4. Приведите пример использования вложенного цикла на практике.
5. Какие существуют условия выхода из циклов и как они реализуются на практике?

Самостоятельная работа № 7 (2 часа)

Тема: Массивы в языке С.
Цель работы: Изучение основ работы с массивами в языке С

Краткие сведения из теории

Массивы представляют собой коллекции однотипных данных, которые хранятся в непрерывной области памяти. Они позволяют организовать и структурировать данные, обеспечивая эффективный доступ к элементам по индексу.
Одномерные массивы
Одномерные массивы — это просто последовательности данных. Каждый элемент может быть доступен по индексу, который начинается с 0.
Пример объявления и инициализации:

```
int numbers[5] = {1, 2, 3, 4, 5};
```

Двумерные массивы
Двумерные массивы можно рассматривать как таблицы, где данные организованы в строки и столбцы.
Пример объявления и инициализации:

```
int matrix[3][3] = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9}
};
```

Альтернативный синтаксис объявления массивов
Помимо стандартного способа объявления массивов, можно использовать альтернативные подходы, такие как указатели и динамическое выделение памяти с помощью malloc.
Пример альтернативного объявления:

```
int* dynamicArray;
dynamicArray = (int*)malloc(sizeof(int) * 5);
```

Получение длины массива и элементов массива
Для статически выделенных массивов длину можно получить, разделив общий размер массива на размер одного элемента:

```
int length = sizeof(numbers) / sizeof(numbers[0]);
```

Для динамически выделенных массивов необходимо отслеживать размер массива вручную, так как стандартные функции С не предоставляют прямого способа получения длины динамических массивов.

Задания для выполнения студентами

1. Одномерные массивы:
    - Напишите программу, которая создает одномерный массив из 10 чисел, заполняет его случайными значениями и выводит на экран.
2. Двумерные массивы:
    - Создайте программу, которая объявляет двумерный массив 4x4, заполняет его случайными значениями и выводит на экран в виде таблицы.
3. Альтернативный синтаксис:
    - Используя malloc, создайте динамический массив из 5 элементов, заполните его значениями, введенными пользователем, и выведите эти значения.
4. Длина массива:
    - Напишите программу, которая создает массив из 5 чисел и выводит его длину с использованием подхода из теоретической справки.
5. Работа с элементами массива:

- Реализуйте программу, которая принимает ввод пользователя (например, числа от 1 до 100) и добавляет их в массив до тех пор, пока не будет введено число 0. Затем отобразите все введенные числа.

Контрольные вопросы

1. В чем разница между одномерными и двумерными массивами?
2. Как можно получить длину статического массива в языке C?
3. Почему важно отслеживать размеры динамически выделяемых массивов?
4. Поясните, как работать с элементами массива, используя индексы.
5. Какие преимущества и недостатки есть у статических и динамических массивов?

Самостоятельная работа № 8 (2 часа)

Тема: Математические вычисления в языке С.
Цель работы: Освоить основы математических вычислений, округления чисел и генерации случайных чисел в языке С.

Краткие сведения из теории

Язык С поддерживает разнообразные математические операции, включая сложение, вычитание, умножение и деление. Основные математические функции находятся в стандартной библиотеке math.h, которая предоставляет такие функции, как:

- pow(): возведение в степень
- sqrt(): извлечение квадратного корня
- sin(), cos(), tan(): тригонометрические функции

Округление чисел
Округление чисел в С может выполняться с помощью различных функций:

- ceil(): округляет число вверх до ближайшего целого
- floor(): округляет число вниз до ближайшего целого
- round(): округляет число до ближайшего целого (в сторону четного)
- trunc(): отбрасывает дробную часть числа

Примеры использования:

```
#include <math.h>
double num = 3.7;
double rounded = round(num); // 4
```

Генерация случайных чисел
Для генерации случайных чисел в С используется функция rand(), которая возвращает случайное целое число. Для получения более разнообразных значений можно использовать srand(), чтобы задать начальное значение (seed). Это позволяет получать разные последовательности случайных чисел при каждом запуске программы.
Пример:

```
#include <stdlib.h>
#include <time.h>
srand(time(NULL)); // установка начального значения
int randomNum = rand() % 100; // случайное число от 0 до 99
```

Задания для выполнения студентами

1. Основные математические операции:
    - Напишите программу, которая запрашивает у пользователя два числа и выполняет с ними все основные арифметические операции (сумма, разница, произведение, частное). Выводите результат на экран.
2. Округление чисел:
    - Создайте программу, которая запрашивает у пользователя десятичное число и выводит его округленное значение с использованием функций ceil(), floor(), round() и trunc().
3. Генерация случайных чисел:
    - Разработайте программу, которая генерирует 10 случайных чисел в диапазоне от 1 до 100 и выводит их на экран. Используйте srand() для установки начального значения.
4. Случайные числа с округлением:
    - Напишите программу, которая генерирует 5 случайных чисел с плавающей точкой в диапазоне от 0 до 10, округляет каждое число с помощью round() и выводит результат.
5. Статистика случайных чисел:

- Реализуйте программу, которая генерирует 100 случайных целых чисел в диапазоне от 1 до 50, сохраняет их в массив и затем вычисляет и выводит среднее значение и максимальное значение сгенерированных чисел.

Контрольные вопросы

1. Какие функции в C используются для округления чисел и как они различаются по своему поведению?
2. Как работает функция rand() и как можно вводить случайные числа в программу?
3. Что происходит при использовании srand() и почему это важно?
4. Объясните, как использовать стандартную библиотеку math.h для выполнения математических операций.
5. Почему желательно использовать генерацию случайных чисел с установкой начального значения (seed)?

Самостоятельная работа № 9 (2 часа)

Тема: Символы и строки в языке С.
Цель работы: Изучение основ обработки символов и строк в языке С.

Краткие сведения из теории

В языке С символы и строки представляют собой последовательности символов, которые обрабатываются с помощью массивов типа char.

- Символ: Один символ, который хранится в переменной типа char.

Пример:

```
char letter = 'A';
```

- Строка: Массив символов, заканчивающийся нуль-символом ('\0'). Строки считываются и записываются как массивы, и каждая строка должна содержать один дополнительный элемент для указания конца.

Пример:

```
char str[20] = "Hello, World!";
```

Стандартные функции для обработки строк

Стандартная библиотека С (string.h) предоставляет множество функций для работы со строками:

- strlen(): возвращает длину строки.
- strcpy(): копирует одну строку в другую.
- strcat(): соединяет две строки.
- strcmp(): сравнивает две строки.
- strchr(): находит первое вхождение символа в строку.

Пример использования:

```
#include <string.h>
char str1[20] = "Hello";
char str2[20] = "World";
strcat(str1, str2); // str1 теперь содержит "HelloWorld"
```

Обработка символов

Для работы с отдельными символами можно использовать преобразования и проверки, например:

- isdigit(): проверяет, является ли символ числом.
- isalpha(): проверяет, является ли символ буквой.
- toupper(): преобразует символ в верхний регистр.
- tolower(): преобразует символ в нижний регистр.

Пример:

```
#include <ctype.h>
char c = 'a';
char upper = toupper(c); // upper теперь содержит 'A'
```

Задания для выполнения студентами

1. Сравнение строк:
    - Напишите программу, которая запрашивает у пользователя две строки и сравнивает их с помощью функции strcmp(). Выведите на экран, являются ли строки одинаковыми.
2. Длина строки:
    - Разработайте программу, которая запрашивает у пользователя строку и выводит её длину, используя функцию strlen().
3. Копирование строк:
    - Создайте программу, которая копирует строку, введенную пользователем, в другую строку с использованием strcpy(), а затем выводит обе строки на экран.

4. Соединение строк:
    - Напишите программу, которая запрашивает у пользователя две строки и соединяет их в одну, используя strcat(), а затем выводит получившуюся строку.
5. Обработка символов:
    - Реализуйте программу, которая запрашивает у пользователя строку и подсчитывает количество букв, цифр и пробелов, а также выводит строку в верхнем регистре.

Контрольные вопросы

1. Чем строка в C отличается от строки в других языках программирования, например, в Python или Java?
2. Какие функции предоставляются стандартной библиотекой string.h для работы со строками?
3. Как обрабатываются строки в C и почему они требуют специального обозначения конца (нуль-символа)?
4. Объясните, как использовать функции для обработки символов, такие как isdigit() и isalpha().
5. Почему важно правильно управлять памятью при работе со строками и символами в языке C?

Самостоятельная работа № 10 (2 часа)

Тема: Поиск, сравнение при обработке символов и строк.
Цель работы: Углубить знания в области поиска и сравнения строк и символов в языке С.

Краткие сведения из теории

В языке С строки представляют собой массивы символов, которые заканчиваются нуль-символом (\0). Для работы с ними используются функции из стандартной библиотеки string.h, которые предоставляют средства для поиска и сравнения строк.
Основные функции для поиска строк

- strchr(): ищет первое вхождение заданного символа в строке.

Пример:

```
char *str = "Hello, World!";
char *ptr = strchr(str, 'o'); // ptr указывает на первое 'o'
```

- strstr(): ищет первое вхождение подстроки в строке.

Пример:

```
char *haystack = "Hello, World!";
char *needle = "World";
char *result = strstr(haystack, needle); // result указывает на начало "World"
```

Функции для сравнения строк

- strcmp(): сравнивает две строки, возвращая 0, если строки равны; отрицательное значение, если первая строка меньше, и положительное, если больше.

Пример:

```
int result = strcmp("Apple", "Banana"); // result будет положительным
```

- strncmp(): сравнивает первые n символов двух строк.

Пример:

```
int result = strncmp("Hello", "Hello, World!", 5); // result будет равен 0
```

Задания для выполнения студентами

1. Поиск символа в строке:
    - Напишите программу, которая запрашивает у пользователя строку и символ, а затем использует strchr() для поиска этого символа в строке. Выведите позицию первого вхождения символа. Если символ не найден, выведите соответствующее сообщение.
2. Поиск подстроки:
    - Создайте программу, которая запрашивает у пользователя строку и подстроку. Используйте strstr() для поиска подстроки в строке и выведите позицию её первого вхождения. Если подстрока не найдена, выведите сообщение об этом.
3. Сравнение строк:
    - Напишите программу, которая запрашивает у пользователя две строки и сравнивает их с помощью strcmp(). Выведите на экран, какая строка больше или если строки равны.
4. Частичное сравнение строк:
    - Реализуйте программу, которая запрашивает у пользователя две строки и количество символов для сравнения n. Используйте strncmp() для сравнения первых n символов обеих строк и выведите результат.
5. Подсчет вхождений:
    - Напишите программу, которая запрашивает строку и символ, а затем подсчитывает, сколько раз символ встречается в строке. Используйте strchr() для выполнения этой задачи.

Контрольные вопросы

1. Как strchr() и strstr() различаются по своему назначению и использованию?

2. Что возвращает функция strcmp(), и как интерпретировать её результат?
3. В чем разница между strcmp() и strncmp()?
4. Как работает поиск символа в строке с помощью strchr()? Приведите пример.
5. Почему важно правильно обрабатывать строки в C, и какие потенциальные ошибки могут возникнуть при работе с ними?

Самостоятельная работа № 11 (2 часа)

Тема: Обзор основных принципов ООП.
Цель работы: Изучить ключевые принципы объектно-ориентированного программирования.

Краткие сведения из теории

Объектно-ориентированное программирование (ООП) представляет собой парадигму программирования, основанную на использовании объектов и классов. Основные принципы ООП включают:

1. Инкапсуляция:
    - Это процесс объединения данных (атрибутов) и методов (функций), работающих с данными, в одну логическую единицу — объект. Инкапсуляция позволяет скрыть внутреннее состояние объекта и защищает его от неконтролируемого доступа.
2. Наследование:
    - Это механизм, позволяющий создавать новый класс на основе уже существующего, что обеспечивает переиспользование кода. Новый класс (наследник) наследует атрибуты и методы родительского класса, а также может добавлять свои собственные.
3. Полиморфизм:
    - Это способность объектов разных классов обрабатывать одно и то же сообщение (вызов метода) по-разному. Полиморфизм позволяет использовать интерфейсы и абстрактные классы, что делает код более гибким и расширяемым.
4. Абстракция:
    - Этот принцип позволяет выделить основные характеристики объекта и игнорировать несущественные детали. Абстракция помогает создать упрощённые модели реального мира, фокусируясь на наиболее значимых аспектах.

Задания для выполнения студентами

1. Инкапсуляция:
    - Напишите класс на выбранном вами языке программирования, который будет представлять сущность "Книга". В классе должно быть несколько приватных атрибутов (например, название, автор, год издания) и методы для их получения и изменения.
2. Наследование:
    - Создайте базовый класс "Транспортное средство" с общими характеристиками (скорость, цвет), и два дочерних класса: "Автомобиль" и "Велосипед". Реализуйте методы, которые будут показывать уникальные характеристики каждого транспортного средства.
3. Полиморфизм:
    - Реализуйте интерфейс "Форма" с методом площадь(). Создайте два класса, "Квадрат" и "Круг", которые реализуют этот интерфейс, и напишите код, который вычисляет площадь каждой формы, используя полиморфизм.
4. Абстракция:
    - Создайте абстрактный класс "Животное" с абстрактным методом громкость_голоса(). Реализуйте два класса "Собака" и "Кошка", которые наследуют от этого класса и дают свои реализации метода.
5. Практическая задача:

- Разработайте простую игру "Угадай число", в которой используются принципы ООП. Класс "Игра" должен управлять логикой игры, а класс "Игрок" будет представлять игрока с возможностью ввода числа и получения подсказок.

Контрольные вопросы

1. Что такое инкапсуляция и как она помогает в разработке программ?
2. Как наследование позволяет избегать дублирования кода?
3. Объясните, как работает полиморфизм и приведите его пример на практике.
4. Что такое абстракция и какую роль она играет в ООП?
5. Как принципы ООП способствуют более структурированному и удобочитаемому коду?

Самостоятельная работа № 12 (2 часа)

Тема: Понятие класса.
Цель работы: Понять концепции классов и экземпляров.

Краткие сведения из теории

Класс — это шаблон или чертеж, который определяет свойства (атрибуты) и поведения (методы) объектов, создаваемых на его основе. Он является основой объектно-ориентированного программирования и служит для организации данных и функций в логически связанные структуры.
Пример объявления класса на Python:

```
class Car:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year

    def display_info(self):
        return f"{self.year} {self.brand} {self.model}"
```

Экземпляр класса
Экземпляр класса (или объект) — это конкретная реализация класса, содержащая специфические значения для его атрибутов. Создание экземпляра класса дает доступ ко всем методам и свойствам, определенным в этом классе.
Пример создания экземпляра:

```
my_car = Car("Toyota", "Camry", 2020)
print(my_car.display_info())  # Вывод: 2020 Toyota Camry
```

Примеры классов и экземпляров класса на языке Java
и C.
1. Класс "Студент"

```
class Student {
    private String name;
    private int age;
    private String specialty;

    // Конструктор
    public Student(String name, int age, String specialty) {
        this.name = name;
        this.age = age;
        this.specialty = specialty;
    }

    // Метод для отображения информации о студенте
    public String displayInfo() {
        return "Имя: " + name + ", Возраст: " + age + ", Специальность: " + specialty;
    }

    // Метод для проверки возраста
    public void checkAge() {
        if (age >= 18) {
            System.out.println(name + " старше 18 лет.");
```

```
        } else {
            System.out.println(name + " младше 18 лет.");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Student student1 = new Student("Иван", 20, "Информатика");
        Student student2 = new Student("Алексей", 17, "Математика");

        System.out.println(student1.displayInfo());
        student1.checkAge();

        System.out.println(student2.displayInfo());
        student2.checkAge();
    }
}
```

Примеры на C

В C нет встроенной поддержки классов, но можно создать структуры, которые выполняют аналогичные функции. Вот пример:

Класс "Студент" с использованием структуры

```
#include <stdio.h>
#include <string.h>

// Определение структуры для студента
struct Student {
    char name[50];
    int age;
    char specialty[30];
};

// Функция для отображения информации о студенте
void displayInfo(struct Student student) {
    printf("Имя: %s, Возраст: %d, Специальность: %s\n", student.name, student.age,
student.specialty);
}

// Функция для проверки возраста
void checkAge(struct Student student) {
    if (student.age >= 18) {
        printf("%s старше 18 лет.\n", student.name);
    } else {
        printf("%s младше 18 лет.\n", student.name);
    }
}

int main() {
    // Создание экземпляров структур
    struct Student student1;
    strcpy(student1.name, "Иван");
    student1.age = 20;
    strcpy(student1.specialty, "Информатика");
```

```
    struct Student student2;
    strcpy(student2.name, "Алексей");
    student2.age = 17;
    strcpy(student2.specialty, "Математика");

    // Вывод информации о студентах
    displayInfo(student1);
    checkAge(student1);

    displayInfo(student2);
    checkAge(student2);

    return 0;
}
```

Задания для выполнения студентами

1. Объявление класса:
    - Создайте класс "Студент", который содержит атрибуты: имя, возраст и специальность. Реализуйте методы для отображения информации о студенте и изменения специальности.
2. Создание экземпляров:
    - Напишите программу, в которой создается несколько экземпляров класса "Студент". Выведите информацию о каждом студенте на экран.
3. Класс с методами:
    - Расширьте класс "Студент", добавив метод, который будет проверять, если студент старше 18 лет, и выводить соответствующее сообщение.
4. Класс с конструктором:
    - Создайте класс "Книга", определите его атрибуты (название, автор, год издания), используя конструктор. Реализуйте метод для отображения информации о книге.
5. Практическое задание:
    - Разработайте класс "Компания" с атрибутами (название, основной вид деятельности) и методом, который будет добавлять сотрудников (экземпляры класса "Сотрудник") в базу данных компании.

Контрольные вопросы

1. Что такое класс и каковы его основные компоненты?
2. В чем разница между классом и экземпляром класса?
3. Как объявляется класс в выбранном вами языке программирования?
4. Для чего используется метод __init__() в Python?
5. Как вы можете получить доступ к атрибутам экземпляра класса?

Самостоятельная работа № 13 (2 часа)

Тема: Модификаторы.
Цель работы: Изучить концепции модификаторов доступа и их применение в объектно-ориентированном программировании

Краткие сведения из теории

Модификаторы доступа в объектно-ориентированных языках программирования (например, Java) определяют уровень видимости классов, методов и атрибутов. Основные модификаторы доступа:

1. public:
    - Члены класса доступны из любого места программы.
2. protected:
    - Члены класса доступны в пределах своего пакета и в подклассах, даже если они находятся в других пакетах.
3. default (package-private):
    - Члены класса доступны только в пределах своего пакета. Если модификатор не указан, используется этот уровень доступа по умолчанию.
4. private:
    - Члены класса доступны только внутри данного класса, что обеспечивает инкапсуляцию и защиту данных.

Модификатор final
Модификатор final используется для объявления классов, методов и переменных с намерением запретить их дальнейшие изменения:

- Final Class: Класс не может быть наследован.
- Final Method: Метод не может быть переопределен в подклассах.
- Final Variable: Переменная становится константой и не может быть изменена после инициализации.

Модификатор static
Модификатор static позволяет создать свойства и методы класса, которые принадлежат самому классу, а не его экземплярам. Это означает, что они могут быть использованы без создания объекта класса:

- Static Variable: Переменная соответствует одному экземпляру для всех объектов.
- Static Method: Метод может быть вызван без создания объекта класса.

Задания для выполнения студентами

1. Использование модификаторов доступа:
    - Создайте класс Employee с атрибутами name, salary и методами для изменения и получения значений атрибутов. Определите разные уровни доступа для каждого атрибута (public, protected, private).
2. Применение модификатора final:
    - Создайте класс MathConstants, в котором объявите константы для математических значений, таких как PI и E. Обозначьте эти значения как final. Попробуйте переопределить одно из значений (подскажите, что это должно вызвать ошибку компиляции).
3. Применение модификатора static:
    - Создайте класс Counter, в котором будет один статический атрибут count и методы для увеличения и получения значения count. Проверьте, что общее количество созданных экземпляров класса корректно обновляется, даже если они создаются через разные экземпляры класса.
4. Смешанное использование:

- Создайте класс Circle с полем radius (private) и статическим методом calculateArea для расчета площади круга. Метод должен принимать радиус как параметр, а не быть привязанным к конкретному объекту.

Контрольные вопросы

1. Что определяет уровень доступа, установленный модификатором private?
2. Какой модификатор доступа позволит наследовать члены класса только в его подклассах?
3. В чем разница между final переменной и обычной переменной?
4. Как модификатор static влияет на переменные и методы в классе?
5. Почему может быть полезно использовать статические методы и переменные?
6. Приведите пример, когда использование final для метода может быть оправдано.

Самостоятельная работа № 14 (2 часа)

Тема: Процессы.
Цель работы: Углубить свои знания о пакетах, директивах импорта и управлении путями для компиляции

Краткие сведения из теории

Пакеты в Java представляют собой группы связанных классов и интерфейсов, которые упрощают организацию и управление кодом. Использование пакетов помогает избежать конфликтов имен, улучшить структуру проекта и упростить процесс его поддержки.
Создание пакетов: Чтобы создать пакет, используйте директиву package в начале файла.
Например:

```
package com.example.myapp;
```

Директивы импорта
Директивы импорта позволяют использовать классы из других пакетов без необходимости полностью указывать их имена. Директива импорта указывается в начале файла программы.
Пример:

```
import java.util.List; // Импорт конкретного класса
import java.util.*;    // Импорт всех классов из пакета
```

Переменная среды CLASSPATH
CLASSPATH — это переменная среды, указывающая JVM (Java Virtual Machine) на места, где она может находить классы и пакеты. Это может включать каталоги, JAR-файлы и ZIP-архивы.
Установка CLASSPATH:

- В Windows:

```
set CLASSPATH=.;C:\path\to\your\classes
```

- В Unix/Linux:

```
export CLASSPATH=.:/path/to/your/classes
```

Задания для выполнения студентами

1. Создание пакета:
    - Создайте пакет com.myapp.utilities. В этом пакете создайте класс MathUtils с методами для выполнения основных математических операций (сложение, вычитание, умножение, деление).
2. Использование директив импорта:
    - Создайте другой класс MainApp в пакете com.myapp и импортируйте MathUtils. Используя этот класс, выполните несколько математических операций и выведите результаты на экран.
3. Работа с переменной CLASSPATH:
    - Напишите инструкции для установки переменной среды CLASSPATH на вашем компьютере. Затем создайте небольшой проект с несколькими пакетами и убедитесь, что ваш проект компилируется и запускается корректно.
4. Создание JAR-файла:
    - Создайте JAR-файл из ваших классов, используя jar команду. Убедитесь, что JAR-файл может быть использован в других проектах, настроив CLASSPATH.
5. Работа с библиотеками:
    - Найдите публичную библиотеку (например, Apache Commons), загрузите ее JAR-файл и добавьте его в ваш проект, настроив CLASSPATH для использования классов из этой библиотеки.

Контрольные вопросы

1. Что такое пакет в Java и как он помогает в организации кода?
2. Какую роль играют директивы импорта в Java?
3. Как вы можете установить переменную среды CLASSPATH, и зачем она нужна?

4. Каковы основные команды для создания и работы с JAR-файлом?
5. В чем отличие между импортом конкретного класса и импортом всех классов из пакета?
6. Как можно разрешить конфликт имен классов, находящихся в разных пакетах?

Самостоятельная работа № 15 (2 часа)

Тема: Расширение и инкапсуляция свойств класса.
Цель работы: Изучить концепции инкапсуляции и расширения свойств классов

Краткие сведения из теории

Инкапсуляция — это принцип объектно-ориентированного программирования, который подразумевает скрытие внутреннего состояния объекта и предоставление доступа к нему только через методы класса. Это достигается с помощью модификаторов доступа (public, private, protected), что обеспечивает безопасность данных и упрощает сопровождение кода.
Примеры инкапсуляции:

- Private атрибуты: Они недоступны снаружи класса, но могут быть изменены через публичные методы (геттеры и сеттеры).

```
public class Person {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Расширение класса
Расширение класса осуществляется с помощью наследования, что позволяет создавать новые классы на основе существующих. Новый класс (подкласс или производный класс) наследует свойства и методы базового класса, что способствует повторному использованию кода и упрощает организацию.
Пример расширения класса:

```
public class Employee extends Person {
    private double salary;

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```

Задания для выполнения студентами

1. Создание класса с инкапсуляцией:
    - Создайте класс Car, который будет иметь свойства: make, model, и year. Все свойства должны быть приватными. Реализуйте геттеры и сеттеры для этих свойств.
2. Наследование и расширение:
    - Создайте класс ElectricCar, который наследует от класса Car. Добавьте новое свойство batteryCapacity, а также методы для его получения и установки.
3. Использование инкапсуляции:

- В классе Car реализуйте метод displayInfo, который выводит всю информацию о машине. Убедитесь, что этот метод может быть использован только через созданный объект.

4. Создание класса Main:
    - Напишите класс Main, в котором создайте объекты Car и ElectricCar, задайте им значения через сеттеры и выведите информацию через метод displayInfo.
5. Обсуждение инкапсуляции и расширения:
    - Напишите краткое описание (1-2 страницы) о том, как инкапсуляция и расширение помогают повысить качество программного обеспечения и обеспечить его простоту в сопровождении.

Контрольные вопросы

1. Что такое инкапсуляция и как она способствует защите данных?
2. Какие преимущества дает использование наследования при проектировании программ?
3. Вызываются ли методы родительского класса в дочернем классе? Как это можно сделать?
4. Чем отличается модификатор доступа protected от private?
5. Почему следует использовать геттеры и сеттеры вместо прямого доступа к полям класса?
6. Приведите пример, когда наследование может привести к сложностям и как их можно избежать.

Самостоятельная работа № 16 (2 часа)

Тема: Наследование как механизм повторного использования кода.
Цель работы: Изучить концепции наследования, его применение и влияние на проектирование программного обеспечения

Краткие сведения из теории

Наследование — это один из основных принципов объектно-ориентированного программирования, который позволяет создавать новые классы (подклассы) на основе существующих (базовых классов). Это позволяет повторно использовать код, устранять дублирование и облегчает процесс управления и расширения программ.
Основные характеристики наследования:

1. Повторное использование кода: Подклассы наследуют свойства и методы базовых классов, что позволяет экономить время на написание повторяющегося кода.
2. Иерархическая структура: Наследование создает иерархию, где базовые классы могут быть обобщениями, а подклассы — более конкретными реализациями.
3. Полиморфизм: Позволяет использовать объекты разных подклассов через ссылку на базовый класс, что упрощает работу с множеством объектов.

Пример:

```
class Animal {
    void eat() {
        System.out.println("Animal is eating");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Dog is barking");
    }
}
```

В этом примере класс Dog наследует метод eat из класса Animal, а также может иметь свои собственные методы.

Задания для выполнения студентами

1. Создание иерархии классов:
    - Создайте базовый класс Vehicle с методами start и stop. Затем создайте подклассы Car и Truck, которые наследуют от Vehicle, добавьте специфичные методы, такие как loadCargo для грузовика.
2. Иерархия животных:
    - Создайте базовый класс Animal и подклассы Cat и Dog. Реализуйте общий метод makeSound в базовом классе, который будет переопределен в каждом из подклассов для реализации специфичных звуков.
3. Использование полиморфизма:
    - Напишите функцию, принимающую массив объектов типа Animal и вызывающую метод makeSound для каждого объекта. Создайте несколько объектов Cat и Dog, затем передайте их в функцию.
4. Исключения и обработка:
    - Добавьте в класс Vehicle метод, который выбрасывает исключение, если попытаться запустить транспортное средство, которое уже работает. Обработайте это исключение в подклассах.
5. Практика реального мира:

- Опишите реальный сценарий, в котором вы можете применить наследование (например, создание приложения для управления библиотекой). Опишите иерархию классов и их методы.

Контрольные вопросы

1. Что такое наследование и какие преимущества оно предоставляет?
2. Как реализовать многократное наследование в Java?
3. Зачем нужно переопределение методов и как оно работает?
4. Как можно использовать полиморфизм в контексте наследования?
5. Какие недостатки возникают при неправильном использовании наследования?
6. Приведите пример, когда целесообразно использовать наследование вместо композиции.

Самостоятельная работа № 17 (2 часа)

Тема: Конструктор при наследовании свойств и методов класса.
Цель работы: Освоить принципы создания и использования пользовательских скриптов. Эта работа поможет студентам глубже понять, как конструкторы функционируют при наследовании, и изучить методы инициализации объектов в объектно-ориентированном программировании, что является важной частью разработки прикладных приложений.

Краткие сведения из теории

Конструктор — это специальный метод, который вызывается при создании нового объекта класса. При наследовании конструкторы работают особым образом: конструктор подкласса может вызывать конструктор базового класса для инициализации унаследованных свойств. Ключевые аспекты работы конструкторов при наследовании:

1. Вызов конструкторов: Конструктор подкласса всегда должен вызывать конструктор базового класса. Это можно сделать с помощью ключевого слова super() в языках программирования, таких как Java или C#.
2. Наследование полей: Подкласс наследует все доступные (public и protected) поля и методы базового класса, однако конструктор базового класса не может быть вызван явно в подклассе, если он не имеет специального конструктора.
3. Параметры конструктора: Подклассы могут передавать параметры в конструктор базового класса, что позволяет гибко настраивать создание объектов.

Пример в Java:

```
class Animal {
    String name;

    Animal(String name) {
        this.name = name;
    }
}

class Dog extends Animal {
    Dog(String name) {
        super(name); // Вызов конструктора базового класса
    }

    void bark() {
        System.out.println(name + " says Woof!");
    }
}
```

В этом примере класс Dog получает имя от базового класса Animal, используя super(name).

Задания для выполнения студентами

1. Создание классов с конструкторами:
    - Создайте базовый класс Person, который имеет поля name и age. Добавьте соответствующий конструктор. Затем создайте подкласс Student, который добавляет поле studentID и реализует свой конструктор, вызывающий конструктор класса Person.
2. Изучение иерархии классов:
    - Создайте базовый класс Shape с полями color. Создайте подклассы Circle и Rectangle, которые добавляют свои уникальные поля и методы, координируя свои конструкторы с конструктором класса Shape.
3. Комплексные конструктора:

- Добавьте в подкласс Student метод, который выводит всю информацию о студенте, включая данные, унаследованные от Person. Запустите данный метод для проверки правильности инициализации полей.

4. Перегрузка конструкторов:
    - Реализуйте перегрузку конструкторов в классе Person, позволяя инициализировать объекты с разным количеством параметров.
5. Практика реального мира:
    - Напишите класс Vehicle и подклассы Car и Bike, которые инициализируют поля через конструкторы. Опишите, как каждый класс может использовать свой конструктор для создания объектов и как это может помочь в управлении параметрами.

Контрольные вопросы

1. Каковы основные функции конструктора в контексте класса?
2. В чем разница между конструктором базового класса и подкласса?
3. Как можно вызывать конструктор базового класса из подкласса?
4. Что произойдет, если конструктор базового класса не будет вызван в подклассе?
5. Как перегрузка конструкторов помогает в программировании?
6. Приведите пример, когда использование конструктора базового класса критично для корректной работы подкласса.

Самостоятельная работа № 18 (2 часа)

Тема: Виртуальные методы и позднее связывание.
Цель работы: Понять концепции виртуальных методов и позднего связывания.

Краткие сведения из теории

Виртуальные методы — это методы, которые могут быть переопределены в производных классах. Они позволяет обеспечить динамическое связывание во время выполнения программы, что является основным аспектом полиморфизма в объектно-ориентированном программировании.
Ключевые особенности виртуальных методов:

1. Позднее связывание: Вызов метода определяется не по типу ссылки, а по фактическому объекту, на который указывает ссылка. Это позволяет использовать один и тот же интерфейс для разных классов.
2. Переопределение: Производные классы могут предоставлять свою реализацию виртуального метода, что позволяет адаптировать поведение метода к конкретным нуждам класса.
3. Объявление: В языках программирования, таких как C++ и Java, виртуальные методы объявляются специальным образом, например, с использованием ключевого слова virtual или override.

Пример в Java:

```
class Animal {
    void makeSound() {
        System.out.println("Animal makes sound");
    }
}

class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Dog barks");
    }
}
```

В этом примере метод makeSound переопределен в классе Dog, и при вызове этого метода на объекте Dog будет использована его собственная реализация.

Задания для выполнения студентами

1. Создание виртуальных методов:
    - Создайте базовый класс Shape с виртуальным методом draw(). Реализуйте подклассы Circle и Square, переопределяющие этот метод. Проверьте, что при использовании объекта типа Shape будет корректно вызываться метод конкретного подкласса.
2. Использование позднего связывания:
    - Создайте коллекцию объектов различных подклассов (например, Dog, Cat) и вызовите виртуальный метод makeSound(). Запишите, какой метод вызывается для каждого объекта.
3. Добавление дополнительных свойств:
    - Расширьте ваш пример с классом Animal, добавив поле name. Реализуйте метод makeSound(), который будет выводить имя животного вместе со звуком.
4. Создание интерфейса:

- Определите интерфейс Playable с методом play(). Реализуйте классы Piano и Guitar, которые реализуют этот интерфейс и определяют собственные версии метода play().

5. Реализация системы типов:
    - Создайте простой класс Player, который будет принимать в себя массив объектов, реализующих интерфейс Playable. Реализуйте метод, который будет вызывать play() на каждом элементе массива.

Контрольные вопросы

1. Что такое виртуальные методы и как они работают в объектно-ориентированном программировании?
2. В чем отличие виртуальных методов от обычных методов?
3. Как объявляется виртуальный метод в различных языках программирования?
4. Почему позднее связывание является важным для реализации полиморфизма?
5. Как механизм виртуальных методов способствует улучшению гибкости и расширяемости кода?
6. Приведите пример, когда использование виртуальных методов приведёт к улучшению архитектуры приложения.

8. ЗАКЛЮЧЕНИЕ

Методические рекомендации разработаны в соответствии с требованиями рабочей учебной программы МДК 02.03 Разработка прикладных приложений.

Организация самостоятельной работы является ключевым элементом в подготовке высококвалифицированных специалистов в области компьютерных систем. Учебный процесс направлен на развитие у студентов не только теоретических знаний, но и практических навыков.

Студенты обеспечены разнообразными заданиями, которые охватывают различные аспекты работы с прикладными приложениями. Задания включают не только теоретические аспекты, но и практические упражнения, позволяющие студентам применять полученные знания на практике. Студенты стимулируются к активному изучению материала, включению в обсуждения и решение проблем, что можно достигнуть через групповую работу, проектные задания и участие в обсуждениях. Это помогает развить критическое мышление и аналитические способности.

Оценка результатов самостоятельной работы является многогранной, включая как формальные, так и неформальные методы. Используются тесты, рефераты, практические задания и проекты, чтобы обеспечить всестороннюю оценку знаний и навыков студентов.

Регулярная обратная связь со стороны преподавателя позволяет студентам корректировать свои действия и препятствовать появлению ошибок в процессе обучения. Обсуждение выполненных заданий помогает углубить понимание материала и выявить зоны для улучшения.

Таким образом, эффективная организация самостоятельной работы помогает обучающимся не только овладеть необходимыми знаниями и навыками, но и подготовит их к профессиональной деятельности в области компьютерных систем и комплексов. Такой подход создаст фундамент для дальнейшего успешного обучения и профессионального роста студентов.

9. ПРИЛОЖЕНИЕ

1. Требования к оформлению реферата

Объем реферата – 20 – 25 стр. печатного текста. Шрифт – не более 14 pt, TimesNewRoman, интервал – 1,5, поля: верхнее, нижнее, левое – 2 см, правое 1,5 см.

На титульном листе указывается название работы, ФИО студента и группа, ФИО преподавателя (научного руководителя), проверяющего и оценивающего реферат, наименование кафедры и учебного заведения. Тема реферата может быть сформулирована самостоятельно, по согласованию с преподавателем.

Название работы оформляется следующим образом:

Реферат МДК 02.03 Разработка прикладных приложений на тему: «……»

Текст реферата печатается на одной стороне страницы; сноски и примечания печатаются на той же странице, к которой они относятся (через 1 интервал, более мелким шрифтом, чем текст). Основной текст должен сопровождаться иллюстративным материалом (рисунки, фотографии, диаграммы, схемы, таблицы, программы). Если в основной части содержатся цитаты или ссылки на высказывания, необходимо указать номер источника по списку, приведенному в конце реферата, и страницу в квадратных скобках в конце цитаты или ссылки.

Реферат – это краткое изложение в письменной форме содержания прочитанных книг и документов; сообщение об итогах изучения научного вопроса; доклад на определенную тему, освещающий ее вопросы на основе литературных и других источников. Целью написания реферата является углубление знаний по конкретной проблеме, получение навыков работы с научной и научно-популярной литературой. Работа над рефератом требует, как правило, не менее месяца.

В процессе работы над проблемой необходимо:

- вычленить проблему;
- самостоятельно изучить проблему на основе первоисточников;
- дать обзор использованной литературы;
- последовательно и доказательно изложить материал;
- правильно оформить ссылки на источники.

2. Обязательные структурные элементы реферата:

1. Введение, в котором описывается актуальность проблемы, определяются цели и задача реферата; объем введения – 1 - 2 страницы.
2. Содержание.
3. Текст реферата должен содержать:
   - обоснование выбранной темы;
   - сравнительный анализ литературы по проблеме;
   - изложение собственной точки зрения на проблему;
   - выводы и предложения;
   - заключение.
4. Список использованных источников должен оформляться в соответствии с ГОСТом и может содержать не только названия книг, журналов, газет, но и любые источники информации (например, сведения из сети Интернет, информацию из теле- и радиопередач, а также частные сообщения каких-либо специалистов, высказанные в личных беседах их с автором реферата).

Реферат излагается доступным научным (научно-популярным) языком в относительно сжатой форме с использованием облегченных синтаксических конструкций. Такие конструкции могут стать своеобразным планом реферативной статьи: “ В рассматриваемой статье ставится ряд вопросов …Автор подчеркивает, что … Более подробно рассмотрена проблема… Анализируются разные точки зрения … В заключение необходимо отметить что …” и т.д.

При выставлении оценки за реферат учитываются следующие компоненты:

- содержательная часть (глубина проработки проблемы, структура работы, объем проанализированных источников и т.п.);
- оформление (соответствие стандарту, эстетика оформления, наличие иллюстративного материала и т.п.);
- защита реферата (ориентация в тексте реферата, ответы на вопросы и т.п.).

Реферат сдается в отпечатанном виде и на электронном носителе.

3. Темы рефератов

1. Основы объектно-ориентированного программирования: Принципы и подходы к разработке приложений.
2. Использование паттернов проектирования в разработке прикладных приложений: Обзор распространённых паттернов и их применение.
3. Разработка пользовательского интерфейса: Принципы проектирования UI/UX для прикладных приложений.
4. Тестирование программного обеспечения: Подходы и методологии тестирования приложений, включая юнит-тестирование и автоматизированное тестирование.
5. Разработка веб-приложений: Технологии и инструменты, используемые в веб-разработке.
6. Контейнеризация и микросервисы: Как Docker и Kubernetes меняют подходы к разработке и развёртыванию приложений.
7. Разработка на основе облачных технологий: Преимущества и вызовы при использовании облачных платформ.
8. Интеграция API в приложениях: Способы и практики работы с API в разработке приложений.
9. Безопасность приложений: Основные принципы и подходы к обеспечению безопасности программного обеспечения.
10. Мобильные приложения: тренды и технологии: Анализ современных технологий разработки мобильных решений.
11. Архитектура программных систем: Основные подходы к проектированию архитектуры приложений.
12. Инструменты и методологии Agile в разработке приложений: Как Agile преобразует традиционные подходы к разработке ПО.
13. SQL vs NoSQL базы данных: Сравнение и выбор подходящей базы данных для прикладных решений.
14. Разработка приложений с использованием искусственного интеллекта: Потенциал и вызовы.
15. Экологические аспекты разработки программного обеспечения: Влияние IT на окружающую среду и методы минимизации воздействия.
16. Системы управления версиями: Роль Git и других систем в процессе разработки приложения.
17. Машинное обучение в приложениях: Как ML влияет на функциональность современных приложений.
18. Интернет вещей (IoT): Применение IoT в прикладных решениях и их развитие.
19. Реализация DevOps: Методы и инструменты, способствующие интеграции разработки и эксплуатации.
20. Разработка приложений на основе микрофронтендов: Подходы к созданию сложных пользовательских интерфейсов.
21. Кроссплатформенная разработка: Технологии и инструменты для разработки приложений на разных платформах.
22. Использование шаблонов проектирования: Как шаблоны помогают структурировать код и улучшать его сопровождаемость.
23. API-first подход: Разработка приложений с акцентом на API и его преимущества.

24. Состояние и управление состоянием в приложениях: Это подходы к управлению состоянием UI и данных.
25. Системы управления проектами в разработке ПО: Методы и инструменты для эффективной работы команд.
26. Исследование и применение блокчейн-технологий: Как блокчейн может быть использован в прикладных решениях.
27. Адаптивные и реактивные приложения: Принципы и инструменты для создания высокоотзывчивого интерфейса.
28. Роль тестирования в CI/CD: Как тестирование интегрируется в процесс непрерывной интеграции и доставки.
29. Этические аспекты разработки ПО: Проблемы и решения в контексте разработки программного обеспечения.
30. Разработка приложений для последних технологий: Использование AR и VR в приложениях.

10. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Основные источники:

1. Соколова, В. В. Разработка мобильных приложений : учебное пособие для среднего профессионального образования / В. В. Соколова. — Москва : Издательство Юрайт, 2024. — 160 с. — (Профессиональное образование). — ISBN 978-5-534-16868-6. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/542342.

2. Трофимов, В. В. Основы алгоритмизации и программирования : учебник для среднего профессионального образования / В. В. Трофимов, Т. А. Павловская ; под редакцией В. В. Трофимова. — 4-е изд. — Москва : Издательство Юрайт, 2024. — 119 с. — (Профессиональное образование). — ISBN 978-5-534-17498-4. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/539994.

3. Батаев, А. В. Технология прикладного программирования: учебник / А. В. Батаев, Н. Ю. Гагарина, Л. Г. Введение в архитектуру программного обеспечения: учеб. пособие / Л. Г. Гагарина, А. Р. Федоров, П. А. Федоров. - М.: ИД «ФОРУМ: ИНФРА-М», 2017.-320 с.

4. Гагарина, Л. Г. Технология разработки программного обеспечения: учеб. пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул; Под ред. Л. Г. Гагариной. - М.: ИД «ФОРУМ: ИНФРА-М», 2017.-400 с.

5. Культин, Н. Б. C/C++ в задачах и примерах. — 3-е изд., доп. и исправл. — СПб.: БХВ-Петербург, 2019. — 272 с.: ил.

Интернет-ресурсы

www.ttgt.org (Сайт Тихорецкого Техникума Железнодорожного Транспорта)

www.studentlibrary.ru (Электронная библиотека)

www.urait.ru (Электронная библиотека)

www.fcior.edu.ru (Федеральный центр информационно-образовательных ресурсов — ФЦИОР).

www.school-collection.edu.ru (Единая коллекция цифровых образовательных ресурсов).

www.intuit.ru/studies/courses (Открытые интернет-курсы «Интуит» по курсу «Информатика»).

www.ru.iite.unesco.org/publications (Открытая электронная библиотека «ИИТО ЮНЕСКО» по ИКТ в образовании).