САМАРСКИЙ ПОЛИТЕХ
Опорный университет

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Самарский государственный технический университет»
(ФГБОУ ВО «СамГТУ»)

Методические указания по выполнению лабораторных работ по дисциплине:

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

для обучающихся очной, очно-заочной и заочной форм обучения
направления подготовки 09.03.01 Информатика и вычислительная техника
профиль Информатика и вычислительная техника в нефтехимическом производстве

Новокуйбышевск, 2025

Методические указания разработаны на кафедре «Информатика и системы управления» в соответствии с требованиями ФГОС ВО по направлению подготовки **09.03.01 Информатика и вычислительная техника**, утвержденного приказом Министерства образования и науки РФ от № 929 от 19.09.2017 и рабочей программой дисциплины «Технологии программирования».

Методические указания предназначены для обучающихся очной, заочной и очно-заочной форм обучения и содержат указания по выполнению лабораторных работ, а также требования к оформлению отчетов по их выполнению.

Разработчик(и): Рубцова Т.П., Волкодаева А.В.

Содержание

1. Общие положения 4
2. Правила работы в лаборатории 5
3. Алгоритм проведения лабораторной работы 7
4. Материально-техническое обеспечение выполнения лабораторных работ 8
5. Учебно-методическое обеспечение выполнения лабораторных работ 9
6. Содержание лабораторных работ 11
7. Критерии и показатели оценки результата выполнения лабораторных работ 36
Приложение А 38
Приложение Б 39

1. Общие положения

Лабораторные занятия – одна из форм практического занятия, являющаяся эффективной формой учебных занятий в образовательной организации. Лабораторные занятия имеют выраженную специфику в зависимости от учебной дисциплины, углубляют и закрепляют теоретические знания. Лабораторные занятия дают наглядное представление об изучаемых процессах, обучающиеся осваивают постановку и ведение эксперимента, учатся оценивать полученные результаты, делать выводы и обобщения.

Лабораторная работа – это форма организации учебного процесса, в рамках которой обучающиеся по заданию и под руководством преподавателя самостоятельно выполняют специально разработанные задания. Лабораторная работа как вид учебного занятия должна проводиться в специально оборудованных учебных лабораториях. Продолжительность – не менее двух академических часов. Необходимыми структурными элементами лабораторной работы, помимо самостоятельной деятельности обучающегося, являются инструктаж, проводимый преподавателем, а также организация обсуждения итогов выполнения лабораторной работы.

Цели лабораторного занятия:

- углубление и закрепление знания теоретического курса путем практического изучения в лабораторных условиях изложенных в лекциях законов и положений;
- приобретение навыков в научном экспериментировании, анализе полученных результатов;
- формирование первичных навыков организации, планирования и проведения практических работ и исследований.

В зависимости от задач, решаемых на лабораторных занятиях, различают:

- ознакомительные лабораторные занятия, которые проводятся с целью закрепления и конкретизации изученного теоретического материала, а также для изучение конструктивных особенностей, устройство средств производственной деятельности (оборудования, инструментов приспособлений и т.д.) и средств исследовательской деятельности (испытательных установок, приборов и т.д.), а также их наладки и настройки;
- экспериментальные лабораторные занятия, которые проводятся с целью получение новой информации на основе формализованных методов, обеспечивающих накопление знаний, умений и практического опыта и включают экспериментальные и исследовательские задания (по изучению и отработке методики проведения различных исследований, по конструированию, переконструированию и доконструированию различных схем и приспособлений, по исследованию влияния различных факторов на свойства объектов, по определению степени соответствия экспериментальных и расчетных данных, по проверке, иллюстрации, подтверждению законов, закономерностей и т.д.;
- творческие лабораторные занятия (проблемно-поисковые работы), которые ставят соей целью получение новой информации на основе формализованных методов и обеспечивают накопление знаний, умений и практического опыта, а также включают постановку и проведение экспериментов и отличаются они только степенью проблемности экспериментальных задач, при этом речь идет об уровнях проблемности этих задач: новизне объектов, условий, в которых про-водится эксперимент по сравнению с известными ранее (к этой группе лабораторных работ относятся и работы по про-верке различных гипотез учебного и научного уровня проблемности).

При проведении лабораторных занятий учебная группа может делиться на подгруппы численностью не менее 8 человек, а в случае индивидуальной подготовки и менее.

2. Правила работы в лаборатории

Правила работы в лаборатории обязательны для исполнения всеми обучающимися, преподавателями и сотрудниками, находящимися в лаборатории. Нарушение правил влечет за собой предупреждение, отстранение от работы и/или другие дисциплинарные меры, предусмотренные уставом образовательной организации. Администрация лаборатории не несет ответственности за несчастные случаи, произошедшие в результате несоблюдения настоящих правил.

Одежда и защитные средства при выполнении лабораторной работы

При проведении лабораторных работ по дисциплине «Технологии программирования» не предусмотрены специальная одежда и защитные средства. Обучающимся запрещается находиться в аудитории в верхней одежде при работе за компьютером.

Инструктаж перед выполнением лабораторной работы

Перед началом выполнения лабораторной работы обучающиеся должны пройти инструктаж по работе в лаборатории, оснащенной персональными компьютерами. Инструктаж может быть как общим (в начале семестра), так и индивидуальным (перед каждой работой, при необходимости). Инструктаж включает:

1. Общие правила работы в лаборатории.
2. Меры безопасности при работе с компьютерной техникой (электробезопасность, правильная посадка за рабочим местом, перерывы).
3. Правила работы с программным обеспечением.
4. Правила хранения и использования данных (конфиденциальность, резервное копирование).
5. Подтверждение прохождения инструктажа – подпись обучающегося в журнале (при необходимости).

1. Общие правила работы в лаборатории:

- поддерживать чистоту и порядок на рабочем месте;
- не оставлять личные вещи на проходах и рабочих столах;
- не употреблять пищу и напитки за компьютерами;
- не использовать постороннее программное обеспечение без разрешения преподавателя;
- сообщать о любых неисправностях оборудования или программного обеспечения лаборанту или преподавателю.
- соблюдать правильную осанку при работе за компьютером;
- регулярно делать перерывы для отдыха глаз и разминки.

2. Меры безопасности при работе с компьютерной техникой:

- запрет на эксплуатацию поврежденного оборудования: необходимо немедленно прекратить использование компьютера, если имеются повреждения корпуса или силовых кабелей, а также если в розетке отсутствует заземление;
- избежание посторонних предметов на системном блоке: на корпусе системного блока не должно быть посторонних предметов, так как это может привести к вибрациям и сбоям в работе оборудования;
- работа в сухих условиях: следует избегать работы с компьютером условиях повышенной влажности или при открытом корпусе;
- правильное расположение проводов и кабелей: провода должны располагаться так, чтобы исключить риск наступления на них или перегрузки тяжелыми предметами;

- не пытаться самостоятельно ремонтировать компьютерную технику;
- использовать средства защиты от излучения монитора (при необходимости);
- не перекрывать вентиляционные отверстия на системном блоке и мониторе;
- не прикасаться к экрану и корпусу монитора, а также изменять местоположение системного блока и монитора.

3. Правила работы с программным обеспечением:
- необходимо использовать только лицензионное программное обеспечение;
- запрещается устанавливать программное обеспечение без разрешения преподавателя;
- запрещается модернизировать или изменять параметры (настройки) операционной системы.

4. Правила хранения и использования данных
- при необходимости создавать резервные копии своих данных;
- не распространять персональные данные без разрешения;
- не посещать сайты, содержащие вирусы или вредоносное программное обеспечение.

Меры безопасности при выполнении лабораторной работы

При выполнении лабораторной работы обучающиеся обязаны соблюдать следующие меры безопасности:
- включать, отключать, работать за компьютером без разрешения преподавателя или лаборанта;
- подключать и отключать любые периферийные устройства, за исключением флэш-накопителей; прикасаться к соединительным кабелям и их разъемам.
- прикасаться к соединительным кабелям и их разъемам;
- открывать корпус системного блока, монитора, периферийных устройств;
- размещать какие-либо предметы (тетради, дискеты, книги и др.) на элементах оборудования персонального компьютеру;
- продолжать работу при наличии сбоев и неполадок функционирования, нехарактерного шума компьютера или признаков возникновения пожара (запаха гари).
- открывать корпус системного блока, монитора, периферийных устройств;

Перед началом работы обучающиеся должны убедиться в отсутствии видимых повреждений на компьютерах (нарушении целостности корпуса, нарушении изоляции проводов, неисправность индикации включения питания, признаки электрического напряжения на корпусе и т. д.), начинать работу с персональным компьютером только по указанию преподавателя.

В процессе работы обучающиеся должны при непроизвольном отключении персонального компьютера, сбоев в работе, нехарактерного шума или запаха гари необходим о немедленно сообщить преподавателю или лаборанту.

По окончании работы необходимо выполнить операцию выхода из системы (компьютер не выключать!), поставить преподавателя в известность об окончании работы с персональным компьютером.

Ответственность обучающихся при выполнении лабораторной работы

Обучающиеся несут ответственность за сохранность оборудования и программного обеспечения, предоставленных для выполнения лабораторной работы. В случае повреждения по вине обучающегося, он обязан возместить ущерб в установленном порядке. Обучающиеся несут ответственность за нарушение конфиденциальности данных.

3. Алгоритм проведения лабораторной работы

Цель проведения лабораторных работ по дисциплине «Технологии программирования» заключается в формировании у обучающихся практических навыков применения технологий программирования, при решении задач профессиональной деятельности, разработки алгоритмов и программ, пригодных для практического применения, использования технологий программирования при оснащении отделов, лабораторий, офисов компьютерным и сетевым оборудованием, тестирования работоспособности программ.

Содержание лабораторных работ определяется требованиями к результатам обучения по дисциплине «Технологии программирования» (таблица 1) в виде умений и навыков в соответствии с **компетенциями**.

Таблица 1

Планируемые результаты обучения по дисциплине «Технологии программирования»

Код и наименование компетенции	**Код и наименование индикатора достижения компетенции**	**Результаты обучения (знать, уметь, владеть, соотнесенные с индикаторами достижения компетенции)**
ОПК-2 Способен понимать принципы работы современных информационных технологий и программных средств, в том числе отечественного производства, и использовать их при решении задач профессиональной деятельности	**ОПК-2.2** Применяет современные информационных технологий и программные средства, в том числе отечественного производства, при решении задач профессиональной деятельности	**Уметь** применять технологии программирования, при решении задач профессиональной деятельности. **Владеть** навыками применения технологий программирования, при решении задач профессиональной деятельности.
ОПК-6 Способен разрабатывать бизнес-планы и технические задания на оснащение отделов, лабораторий, офисов компьютерным и сетевым оборудованием	**ОПК-6.1** Выявляет потребности организации в компьютерном и сетевом оборудовании	**Уметь** использовать технологии программирования при оснащении отделов, лабораторий, офисов компьютерным и сетевым оборудованием. **Владеть** навыками использования технологий программирования при оснащении отделов, лабораторий, офисов компьютерным и сетевым оборудованием.
ОПК-8 Способен разрабатывать алгоритмы и программы, пригодные для практического применения	**ОПК-8.1** Составляет алгоритмы, пишет и отлаживает коды на языке программирования	**Уметь** разрабатывать алгоритмы и программы, пригодные для практического применения. **Владеть** навыками разработки алгоритмов и программ, пригодных для практического применения.
	ОПК-8.3 Тестирует работоспособность программы	**Уметь** тестировать работоспособность программ. **Владеть** навыками тестирования работоспособности программ.

Алгоритм проведения лабораторной работы по дисциплине «Технологии программирования» включает шесть основных этапов. Последовательность и содержание выполнения лабораторной работы представлено в таблице 2.

Таблица 2

Последовательность и содержание выполнения лабораторной работы по дисциплине «Технологии программирования»

Этап	Содержание
1. Подготовка к лабораторной работе	Получение задания от преподавателя (описание задачи, данные, требования к отчету). Изучение теоретического материала (конспекты лекций, учебная литература, методические указания). Ознакомление с используемым программным обеспечением. Подготовка плана выполнения работы.
2. Начало лабораторной работы	Запуск необходимого программного обеспечения. Загрузка предоставленных данных или подготовка данных самостоятельно. Создание новых файлов для хранения результатов работы.
3. Выполнение работы	Выполнение анализа данных в соответствии с заданием. Очистка и подготовка данных. Выбор и применение подходящих методов выполнения поставленных задач. Интерпретация результатов анализа. Фиксация промежуточных результатов. Создание таблиц, графиков, диаграмм для визуализации данных. Формулировка выводов и рекомендаций на основе результатов анализа.
4. Завершение работы	Сохранение всех рабочих файлов. Закрытие программного обеспечения. Удаление временных файлов (при необходимости).
5. Оформление отчета	В отчете должны быть четко сформулированы цель работы, описание использованных методов, результаты анализа (с таблицами, графиками), выводы и рекомендации. Отчет должен быть оформлен в соответствии с требованиями методических указаний. Указание источников данных и использованного программного обеспечения.
6. Защита лабораторной работы	Демонстрация преподавателю результатов работы. Ответы на вопросы по методике анализа, интерпретации результатов и сделанным выводам. Объяснение ограничений использованных методов. Предложения по дальнейшему развитию анализа. Обоснование практической значимости полученных результатов лабораторной работы.

4. Материально-техническое обеспечение выполнения лабораторных работ

Для проведения лабораторных занятий и выполнения лабораторных работ по дисциплине «Технологии программирования», образовательная организация располагает материально-технической базой, соответствующей действующим санитарным и противопожарным правилам и нормам. Помещения оснащены оборудованием персональными компьютерами, программным обеспечением, комплектом мебели для обучающихся и преподавателя, а также другими техническими средствами обучения.

Каждый обучающийся обеспечен индивидуальным неограниченным доступом к электронной информационно-образовательной среде образовательной организации из любой точки, в которой имеется доступ к информационно-телекоммуникационной сети «Интернет», как на территории образовательной организации, так и вне ее.

Компьютерная техника оснащена подключения к информационно-телекоммуникационной сети «Интернет» и обеспечением доступа в электронную информационно-образовательную среду образовательной организации.

Образовательная организация обеспечена необходимым комплектом лицензионного и свободно распространяемого программного обеспечения, в том числе отечественного производства (в соответствии с программой дисциплины) и подлежит обновлению при необходимости.

Оборудование и программное обеспечение, используемое при проведении лабораторных работ по дисциплине «Технологии программирования»:

- программное обеспечение: среда разработки для C++ (например, Code::Blocks, Visual Studio Community) и/или Anaconda 3 (Python), браузер, доступ в ЭИОС и Интернет;
- документация по языку C++/Python, cplusplus.com, cppreference.com.

5. Учебно-методическое обеспечение выполнения лабораторных работ

Учебно-методическое обеспечение выполнения лабораторных работ по дисциплине «Технологии программирования» включает:

1. Перечень литературы по дисциплине.
2. Методические указания по выполнению лабораторных работ.
3. Описание процесса проведения лабораторных работ.

1. Перечень литературы по дисциплине.

Основная литература:

1. Python и анализ данных; ДМК Пресс, 2020. – Режим доступа: https://elib.samgtu.ru/getinfo?uid=els_samgtu||iprbooks||125361.
2. Интеллектуальный анализ данных; Издательский Дом Томского государственного университета, 2020. – Режим доступа: https://elib.samgtu.ru/getinfo?uid=els_samgtu||iprbooks||116889 Электронный ресурс.
3. Методы хранения и анализа данных; Ай Пи Ар Медиа, 2022. – Режим доступа: https://elib.samgtu.ru/getinfo?uid=els_samgtu||iprbooks||119065.
4. Объектно ориентированное программирование на языке Python; СанктПетербургский государственный архитектурно-строительный университет, ЭБС АСВ, 2020. – Режим доступа: https://elib.samgtu.ru/getinfo?uid=els_samgtu||iprbooks||117194.
5. Объектно-ориентированное программирование; Профобразование, Ай Пи Ар Медиа, 2022. – Режим доступа: https://elib.samgtu.ru/getinfo?uid=els_samgtu||iprbooks||118969.
6. Практикум по объектно-ориентированному программированию; Лаборатория знаний, 2020. - Режим доступа: https://elib.samgtu.ru/getinfo?uid=els_samgtu||iprbooks||12254.

Дополнительная литература:

1. Python и анализ данных; Профобразование, 2019. – Режим доступа: http://www.iprbookshop.ru/88752.html.
2. Анализ данных; Университет экономики и управления, 2019. – Режим доступа: https://elib.samgtu.ru/getinfo?uid=els_samgtu||iprbooks||89482 Электронный ресурс
3. Введение в теорию программирования. Объектно-ориентированный подход; Профобразование, 2021. – Режим доступа: https://elib.samgtu.ru/getinfo?uid=els_samgtu||iprbooks||102188.
4. Объектно-ориентированное программирование. В 3-х частях. Ч.1; Омский государственный технический университет, 2021. – Режим доступа: https://elib.samgtu.ru/getinfo?uid=els_samgtu||iprbooks||124850.
5. Основы анализа данных; Санкт-Петербургский государственный университет промышленных технологий и дизайна, 2019. – Режим доступа: https://elib.samgtu.ru/getinfo?uid=els_samgtu||iprbooks||102939.

Доступ обучающихся к ЭР НТБ СамГТУ (elib.samgtu.ru) осуществляется посредством электронной информационной образовательной среды университета и сайта НТБ СамГТУ по логину и паролю.

Методические материалы размещены на сайте филиала ФГБОУ ВО «СамГТУ» в г. Новокуйбышевске, в разделе «Сведения об образовательной организации», подраздел «Образование», таблица «Информация по

образовательным программам» в ячейке «Ссылка на иные компоненты, оценочные и методические материалы, а также в предусмотренных ФЗ от 29.12.2012 № 273-ФЗ «Об образовании в РФ» случаях в виде рабочей программы воспитания, календарного плана воспитательной работы, форм аттестации в виде электронного документа».

2. Методические указания по выполнению лабораторных работ.

Методические указания по выполнению лабораторных работ по дисциплине «Технологии программирования» содержат общие положения, правила работы в лаборатории, алгоритм проведения лабораторной работы, материально-техническое обеспечение выполнения лабораторных работ, учебно-методическое обеспечение выполнения лабораторных работ, содержание лабораторных работ, критерии и показатели оценки результата выполнения лабораторных работ.

3. Описание процесса проведения лабораторных работ

Процесс проведения лабораторных работ по дисциплине «Технологии программирования» направлен на формировании у обучающихся практических навыков в соответствии с указанной выше целью. Процесс проведения лабораторных работ состоит из шести этапов, перечисленных выше. Рассмотрим содержание и ход выполнения работ в соответствии с этапами проведения лабораторной работы.

1. Подготовка к лабораторной работе

Преподаватель предоставляет обучающим описание лабораторной работы, включающее цель работы, постановку задачи, требования к программному обеспечению, требования к отчету по результатам проведения лабораторной работы, критерии оценки (параметры, по которым будет оцениваться выполненная работа).

Обучающиеся выбирают и осваивают программные средства, которые будут использоваться для выполнения работы, знакомятся с интерфейсом, основными функциями и инструментами, примерами выполнения учебных заданий, осуществляют поиск дополнительных материалов и документации для выполнения лабораторных задач. Обучающиеся разрабатывают план выполнения лабораторной работы, включающий определение последовательности действий, распределение времени на выполнение каждого этапа работы, подготовку необходимых ресурсов (данные, программное обеспечение, инструменты).

2. Выполнение лабораторной работы (начало лабораторной работы, выполнение работы, завершение работы.

Обучающиеся собирают необходимые для выполнения лабораторной работы данные из указанных источников, подготавливают данные для анализа, осуществляет преобразование данных, масштабирование данных. В процессе анализа данных обучающиеся применяют выбранные методы для решения поставленной задачи, интерпретируют результаты анализа данных, делая выводы о решении поставленной задачи. Выводы должны быть обоснованы результатами анализа. Обучающиеся оформляют результаты работы в виде таблиц, графиков, диаграмм. Формат представления результатов должен быть понятным и наглядным.

3. Оформление и сдача отчета

Обучающиеся оформляют отчет по лабораторной работе. Отчет включает следующие элементы:

- титульный лист (Приложения А);
- цель работы;
- индивидуальный вариант задания;
- листинг программы (исходный код с комментариями, с указанием структуры и специальной функции);

- результаты выполнения программы (скриншот консоли с выводом данных и результата);
- выводы по лабораторной работе.

Отчет должен быть оформлен в соответствии с требованиями. Текст должен быть четким и лаконичным. Таблицы и графики должны быть подписаны и пронумерованы. Формулы должны быть набраны в редакторе формул. Отчет должен быть предоставлен на проверку преподавателю в установленный срок. Образец отчета по лабораторной работе представлен в Приложении Б.

4. Защита лабораторной работы

Обучающиеся готовятся к защите, повторяя материал и анализируя результаты своей работы. Необходимо быть готовым ответить на вопросы преподавателя, касающиеся цели и задач работы, методики выполнения работы, результатов работы и сформулированных выводов.

В процессе защиты, обучающийся кратко излагает цель, задачи и результаты своей работы, отвечает на вопросы преподавателя, демонстрирует понимание результатов анализа и их практической значимости. Преподаватель оценивает качество выполненной работы, уровень знаний обучающегося и его способность применять эти знания на практике.

Обучающиеся должны активно участвовать в выполнении лабораторных работ, проявлять самостоятельность и инициативу. При анализе данных необходимо проявлять критическое мышление, оценивать достоверность информации и обоснованность выводов.

6. Содержание лабораторных работ

В рамках дисциплины «Технологии программирования» предусмотрено 16 академических часов на выполнение лабораторных работ согласно учебному плану. Лабораторные работы предусмотрены в рамках разделов дисциплины:

- структурное программирование;
- объектно-ориентированное программирование.

Темы лабораторных работ и соответствующие им планируемые результаты обучения представлены в таблице 3.

Таблица 3

Темы лабораторных работ и соответствующие планируемые результаты обучения

Темы лабораторных работ	**Кол-во часов**	**Код индикатора достижения компетенции**	**Планируемые результаты обучения (умения, навыки)**
Раздел 1. Структурное программирование			
1. Работа со структурами	2	ОПК-8.1	**Уметь** разрабатывать алгоритмы и программы, пригодные для практического применения. **Владеть** навыками разработки алгоритмов и программ, пригодных для практического применения.
2. Работа с динамическими структурами (динамическое выделение памяти)	2	ОПК-8.1	**Уметь** разрабатывать алгоритмы и программы, пригодные для практического применения. **Владеть** навыками разработки алгоритмов и программ, пригодных для практического применения.
3. Работа с однонаправленными списками	2	ОПК-8.1	**Уметь** разрабатывать алгоритмы и программы, пригодные для практического применения.

Темы лабораторных работ	Кол-во часов	Код индикатора достижения компетенции	Планируемые результаты обучения (умения, навыки)
			Владеть навыками разработки алгоритмов и программ, пригодных для практического применения.
4. Работа с двунаправленными списками	2	ОПК-8.1	**Уметь** разрабатывать алгоритмы и программы, пригодные для практического применения. **Владеть** навыками разработки алгоритмов и программ, пригодных для практического применения.
Раздел 2. Объектно-ориентированное программирование			
5. Классы (инкапсуляция, наследование, полиморфизм)	2	ОПК-2.2	**Уметь** применять технологии программирования, при решении задач профессиональной деятельности. **Владеть** навыками применения технологий программирования, при решении задач профессиональной деятельности.
		ОПК-8.1	**Уметь** разрабатывать алгоритмы и программы, пригодные для практического применения. **Владеть** навыками разработки алгоритмов и программ, пригодных для практического применения.
6. Работа с операциями (перегрузка операторов)	2	ОПК-6.1	**Уметь** использовать технологии программирования при оснащении отделов, лабораторий, офисов компьютерным и сетевым оборудованием. **Владеть** навыками использования технологий программирования при оснащении отделов, лабораторий, офисов компьютерным и сетевым оборудованием.
7. Наследование и виртуальные функции	2	ОПК-8.1	**Уметь** разрабатывать алгоритмы и программы, пригодные для практического применения. **Владеть** навыками разработки алгоритмов и программ, пригодных для практического применения.
		ОПК-8.3	**Уметь** тестировать работоспособность программ. **Владеть** навыками тестирования работоспособности программ.
8. Работа с библиотеками шаблонов (STL)	2	ОПК-2.2	**Уметь** применять технологии программирования, при решении задач профессиональной деятельности. **Владеть** навыками применения технологий программирования, при решении задач профессиональной деятельности.
		ОПК-8.1	**Уметь** разрабатывать алгоритмы и программы, пригодные для практического применения. **Владеть** навыками разработки алгоритмов и программ, пригодных для практического применения.

Содержание лабораторных работ отражает цель, код и наименование индикатора достижения компетенции, виды работ, время выполнения лабораторной работы, информацию о подготовке к лабораторной работе, общее задание для всех вариантов ответов, индивидуальные варианты заданий, методику выполнения,

содержание отчета (шаблон), выводы, контрольные вопросы (для допуска и защиты).

Лабораторная работа №1. Работа со структурами

Цель: сформировать навык использования структур для агрегации разнотипных данных, моделирующих сущность предметной области (нефтехимическое производство). Научиться: создавать структуры, инициализировать их, передавать в функции, производить над ними операции.

Код и наименование индикатора достижения компетенции: ОПК-8.1 Составляет алгоритмы, пишет и отлаживает коды на языке программирования.

Время выполнения: 2 часа.

Подготовка к работе (повторить):

- понятие структуры в языке C/C++ или именованного кортежа/датакласса в Python;
- способы инициализации структур;
- доступ к полям структуры;
- передача структур в функции по значению и по ссылке (или по ссылке/значению в Python).

Общее задание для всех вариантов:

Разработайте программу, которая:

- определяет структуру Equipment (Оборудование) для описания единицы технологического оборудования. Поля структуры приведены в вашем варианте;
- в функции main() создает массив (std::vector в C++ или list в Python) из 3-5 структур Equipment с данными согласно вашему варианту;
- реализует функцию printEquipment(const Equipment& item) (или print_equipment(item)), которая выводит информацию об одном объекте в удобочитаемом виде;
- реализует функцию для решения задачи, специфичной для вашего варианта (например, поиск по критерию, расчет суммарного показателя, фильтрация);
- выводит исходный список оборудования и результат работы специальной функции.

Индивидуальные варианты заданий:

Структура для всех вариантов имеет базовые поля:

- int id; // Уникальный идентификатор
- std::string name; // Наименование оборудования
- std::string workshop; // Цех/установка
- int year_commissioned; // Год ввода в эксплуатацию

Вариант 1. Насосное оборудование.

Доп. поле в структуре: double flow_rate; // Производительность, м³/ч

Специальная задача: Найти и вывести информацию о насосе с максимальной производительностью.

Пример данных: (1, "Насос центробежный ЦН-500", "Цех №1", 2018, 500.0), (2, "Насос шестеренчатый ШН-200", "Цех №2", 2020, 200.0), ...

Вариант 2. Резервуарный парк.

Доп. поле: double volume; // Объем, м³

Специальная задача: Рассчитать суммарный объем всех резервуаров в указанном цехе (цех задается константой в программе).

Пример данных: (1, "РВС-5000", "Склад ГСМ", 2015, 5000.0), (2, "РВС-10000", "Сырьевой парк", 2010, 10000.0), ...

Вариант 3. Реакторы.

Доп. поле: double temperature_max; // Максимальная рабочая температура, °C
Специальная задача: Вывести список реакторов, чья максимальная температура превышает заданное значение (например, 300 °C).
Пример данных: (1, "Риформинг-реактор Р-1", "Установка Л-35/11", 2019, 520.0), (2, "Гидроочистки реактор", "Установка Г-43", 2017, 420.0), ...

Вариант 4. Теплообменники.

Доп. поле: double heat_transfer_area; // Площадь теплообмена, м²
Специальная задача: Найти среднюю площадь теплообмена всех теплообменников.
Пример данных: (1, "Теплообменник кожухотрубный ТК-100", "Цех №5", 2016, 100.5), (2, "Пластинчатый теплообменник ПТ-50", "Цех №3", 2021, 50.2), ...

Вариант 5. Колонны ректификационные.

Доп. поле: int tray_count; // Количество тарелок
Специальная задача: Найти колонну с минимальным количеством тарелок.
Пример данных: (1, "Колонна К-1", "АВТ-6", 2005, 40), (2, "Колонна стабилизации К-2", "АВТ-6", 2005, 30), ...

Вариант 6. Компрессоры.

Доп. поле: double power; // Мощность, кВт
Специальная задача: Подсчитать количество компрессоров, мощность которых ниже заданного порога (например, 1000 кВт).
Пример данных: (1, "Компрессор поршневой КП-10", "Цех КИП", 2018, 750.0), (2, "Турбокомпрессор ТК-25", "Цех №4", 2022, 2500.0), ...

Вариант 7. Печи трубчатые.

Доп. поле: double efficiency; // КПД, %
Специальная задача: Найти печь с максимальным КПД.
Пример данных: (1, "Печь П-1", "Установка ППВ", 2012, 88.5), (2, "Печь П-2", "Установка ППВ", 2014, 91.2), ...

Вариант 8. Емкости смесительные.

Доп. поле: std::string material; // Материал исполнения (сталь, стекло, полипропилен)
Специальная задача: Вывести список всех емкостей, изготовленных из заданного материала (например, "нерж. сталь").
Пример данных: (1, "Смеситель СМ-1000", "Лаборатория", 2020, "стекло"), (2, "Реактор-смеситель РС-5000", "Цех компаундирования", 2019, "нерж. сталь"), ...

Вариант 9. Приборы КИПиА (датчики давления).

Доп. поле: double pressure_max; // Максимальное измеряемое давление, атм.
Специальная задача: Вывести отсортированный по возрастанию давления список датчиков.
Пример данных: (1, "Датчик давления Metran-100", "Установка АВТ", 2023, 160.0), (2, "Датчик диф. давления Сапфир-22", "Цех №2", 2021, 25.0), ...

Вариант 10. Сепараторы.

Доп. поле: double throughput; // Пропускная способность, т/сутки
Специальная задача: Рассчитать общую пропускную способность всех сепараторов, введенных в эксплуатацию после заданного года (например, 2015).
Пример данных: (1, "Газовый сепаратор ГС-1", "Цех подготовки газа", 2018, 1500.0), (2, "Нефтяной сепаратор НС-2", "Устьевая установка", 2010, 800.0), ...

Методика выполнения:

1. Выберите свой вариант задания (по номеру в журнале или указанию преподавателя).
2. Создайте новый проект в среде разработки.
3. Объявите структуру Equipment с полями, указанными в вашем варианте.
4. Напишите функцию printEquipment.
5. Напишите функцию, решающую специальную задачу вашего варианта

6. В функции main() инициализируйте массив/вектор/список данных (минимум 3 записи). Данные должны быть реалистичными для нефтехимического производства.
7. Протестируйте программу. Убедитесь, что специальная функция работает корректно.
8. Выведите результаты на экран.

Содержание отчета (шаблон):

1. Титульный лист (по форме Приложения А).
2. Цель работы.
3. Индивидуальный вариант задания (скопируйте текст своего варианта).
4. Листинг программы (исходный код с комментариями, особенно укажите, где ваша структура и специальная функция).
5. Результаты выполнения программы (скриншот консоли с выводом ваших данных и результата).
6. Выводы.

Выводы:

1. Какие преимущества дает использование структуры Equipment по сравнению с хранением данных в отдельных массивах (int id[...], string name[...] и т.д.)?
2. Опишите, как в вашей программе реализована передача структуры в функции. Почему был выбран именно этот способ (по значению/по ссылке/по константной ссылке)?
3. Как, по вашему мнению, можно расширить эту структуру для более полного описания оборудования?

Контрольные вопросы (для допуска и защиты):

1. Объясните, что такое композиция данных и как структура реализует этот принцип.
2. В какой области памяти (стек или куча) будет размещен массив ваших структур, созданный как локальная переменная в main()?
3. Предложите, как можно модифицировать программу, чтобы данные об оборудовании загружались из текстового файла. Как при этом должна измениться структура?
4. Почему в реальных проектах для подобных целей часто используют не структуры (struct), а классы (class)? В чем ключевое различие применительно к вашей задаче?

Лабораторная работа №2. Работа с динамическими структурами (динамическое выделение памяти)

Цель: сформировать навыки работы с динамической памятью: выделение и освобождение памяти для структур, создание массивов структур в куче. Освоить принципы управления памятью для повышения гибкости программы.

Код и наименование индикатора достижения компетенции: ОПК-8.1 Составляет алгоритмы, пишет и отлаживает коды на языке программирования

Время выполнения: 2 часа.

Подготовка к работе (повторить):

- операторы new и delete в C++ (или malloc/free в C);
- создание динамического массива;
- передача динамических массивов в функции;
- важность освобождения памяти (утечки памяти).

Общее задание для всех вариантов:

Разработайте программу, которая:

- определяет структуру ChemicalReagent (Химический реагент) с полями согласно вашему варианту;

- запрашивает у пользователя количество реагентов N (от 3 до 5);
- динамически выделяет память под массив из N структур ChemicalReagent;
- в цикле заполняет массив данными, вводимыми с клавиатуры (или инициализированными заранее для теста);
- реализует функцию, решающую задачу вашего варианта, которая работает с динамическим массивом;
- выводит результат;
- обязательно освобождает выделенную динамическую память.

Индивидуальные варианты заданий:

Базовая структура для всех вариантов:
- int id;
- std::string name;
- std::string formula.

Вариант 1. Реагенты-ингибиторы коррозии.

Доп. поле: std::string application_point; // Точка ввода (например, "сырьевой насос", "линия рециркуляции")

Специальная задача: найти и вывести все реагенты, применяемые в заданной точке ввода (точка задается константой в программе).

Пример ввода: (1, "Ингибитор К-1", "C12H25N", "сырьевой насос"), (2, "Ингибитор Амино-5", "RNH2", "линия рециркуляции"), (3, "Ингибитор К-1", "C12H25N", "сырьевой насос")

Вариант 2. Деэмульгаторы.

Доп. поле: double dosage; // Норма расхода, г/т

Специальная задача: рассчитать суммарный расход деэмульгатора (в граммах) для заданного объема нефти V (например, 1000 тонн). Сумма = Σ(dosage[i] * V) для всех реагентов.

Пример ввода: (1, "Деэмульгатор Дипроксамин", "C18H37NO2", 25.5), (2, "Деэмульгатор Сепарол", "Полиоксиалкилен", 30.0), …

Вариант 3. Катализаторы крекинга.

Доп. поле: int activity_level; // Уровень активности (усл. ед., 1-10)

Специальная задача: найти катализатор с минимальным уровнем активности.

Пример ввода: (1, "Катализатор цеолитсодержащий", "Zeolite-Y", 8), (2, "Катализатор аморфный", "SiO2*Al2O3", 5), …

Вариант 4. Абсорбенты (поглотители).

Доп. поле: std::string target_component; // Поглощаемый компонент ("H2S", "CO2", "H2O")

Специальная задача: посчитать, сколько абсорбентов предназначено для удаления заданного компонента (компонент вводится с клавиатуры).

Пример ввода: (1, "Моноэтаноламин", "C2H7NO", "H2S"), (2, "Цеолит", "Na2O*Al2O3", "H2O"), ...

Вариант 5. Стабилизаторы полимеров.

Доп. поле: double max_temperature; // Максимальная температура эффективности, °C

Специальная задача: вывести список стабилизаторов, эффективных выше заданной температуры (например, 150 °C).

Пример ввода: (1, "Стабилизатор Фенозан", "C15H24O3", 180.0), (2, "Ионол", "C29H44O2", 120.0), ...

Вариант 6. Реагенты для очистки сточных вод.

Доп. поле: std::string waste_type; // Тип стоков ("нефтесодержащие", "химические", "бытовые")

Специальная задача: найти и вывести первый реагент в массиве, предназначенный для очистки нефтесодержащих стоков.

Пример ввода: (1, "Коагулянт АК-1", "Al2(SO4)3", "химические"), (2, "Флокулянт Магнафлок", "Полиакриламид", "нефтесодержащие"), ...

Вариант 7. Отдушки (маскирующие запах).

Доп. поле: std::string aroma; // Аромат ("цитрус", "хвоя", "ваниль")

Специальная задача: вывести список всех отдушек с заданным ароматом (аромат задается константой).

Пример ввода: (1, "Отдушка Апельсин", "Сложные эфиры", "цитрус"), (2, "Отдушка Лесная", "Терпены", "хвоя"), ...

Вариант 8. Антиоксиданты для топлив.

Доп. поле: std::string fuel_type; // Тип топлива ("бензин", "дизель", "авиакеросин")

Специальная задача: определить, используется ли антиоксидант для бензина. Если да - вывести его название.

Пример ввода: (1, "Агидол-1", "Фенолы", "бензин"), (2, "Ионол", "C29H44O2", "дизель"), ...

Вариант 9. Реагенты для гидроочистки.

Доп. поле: double sulfur_removal; // Степень удаления серы, %

Специальная задача: найти реагент с максимальной степенью удаления серы.

Пример ввода: (1, "Катализатор Ni-Mo", "NiO*MoO3", 95.5), (2, "Катализатор Co-Mo", "CoO*MoO3", 98.2), …

Вариант 10. Промывочные жидкости (растворители).

Доп. поле: std::string target_contaminant; // Удаляемый загрязнитель ("парафин", "смола", "накипь")

Специальная задача: подсчитать, сколько разных растворителей предназначено для удаления парафина.

Пример ввода: (1, "Растворитель Толуол", "C7H8", "смола"), (2, "Растворитель Керосин", "Смесь УВ", "парафин"), …

Методика выполнения:

1. Объявите структуру ChemicalReagent.
2. Запросите N (количество реагентов). Организуйте проверку ввода.
3. Выделите память: ChemicalReagent* reagents = new ChemicalReagent[N];
4. В цикле заполните массив. Для ввода строк используйте std::cin (или getline после ввода чисел).
5. Реализуйте функцию для вашей специальной задачи (например, findByApplicationPoint).
6. Выведите исходный массив и результат.
7. Освободите память: delete[] reagents;

Содержание отчета (шаблон):

1. Титульный лист.
2. Цель работы.
3. Индивидуальный вариант задания.
4. Листинг программы (особое внимание на места выделения и освобождения памяти).
5. Результаты выполнения (скриншоты: ввод данных, вывод результата).
6. Выводы.

Выводы:

1. Чем принципиально отличается работа с динамическим массивом от работы с обычным (статическим) массивом, как в ЛР1?
2. Что произойдет, если не освободить память с помощью delete[]? Как это может повлиять на долго работающую программу на производственном сервере?
3. Какие преимущества дает динамическое выделение памяти в вашей конкретной задаче (например, если бы количество реагентов было неизвестно на этапе компиляции)?

Контрольные вопросы:

1. В чем разница между операторами new/delete и функциями malloc()/free()? Какой подход является более предпочтительным в C++ и почему?
2. Что такое «висячий указатель»? Как избежать этой ошибки?
3. Что будет, если попытаться освободить память с помощью delete (без []) для массива, выделенного через new[]?
4. Как можно модифицировать программу, чтобы добавлять новые реагенты в массив после его первоначального создания? Какие проблемы возникнут и как их решить (подсказка: realloc в C или new + копирование в C++)?

Лабораторная работа №3. Работа с однонаправленными списками

Цель: освоить принципы организации и алгоритмы работы с однонаправленными линейными списками, научиться: создавать, добавлять, удалять элементы, осуществлять поиск и обход списка, применить список для моделирования последовательности событий или объектов в технологическом процессе.

Код и наименование индикатора достижения компетенции: ОПК-8.1 Составляет алгоритмы, пишет и отлаживает коды на языке программирования

Время выполнения: 2 часа.

Подготовка к работе (повторить):

- понятие узла (Node) списка (данные + указатель на следующий элемент);
- базовые операции: инициализация, добавление в начало/конец, вставка после заданного узла, удаление узла, поиск, вывод всего списка;
- особенности работы с указателями при манипуляциях со списком;
- освобождение памяти всего списка.

Общее задание для всех вариантов:

Разработайте программу, которая:

- определяет структуру Node (узел списка), содержащую данные согласно вашему варианту (структуру TechEvent или TechObject), указатель Node* next на следующий узел;
- определяет функции для работы со списком: void pushBack(Node*& head, ...) - добавление в конец; void printList(const Node* head) - вывод списка; функцию для решения специальной задачи вашего варианта;
- в функции main() создает пустой список, добавляет в него не менее 5 элементов с данными (данные могут быть заранее подготовлены);
- выводит созданный список;
- выполняет специальную задачу и выводит результат;
- корректно освобождает всю память, занятую списком.

Индивидуальные варианты заданий:

Базовые данные (хранятся в узле) для всех вариантов - структура:

- int id; // Уникальный номер события/объекта;
- std::string description; // Краткое описание.

Вариант 1. Последовательность операций технологической смены.

Доп. поле в данных: std::string operator_name; // ФИО оператора

Специальная задача: Найти и вывести все операции, выполненные заданным оператором (ФИО задается константой).

Пример элементов списка:{id: 1, desc: "Пуск насоса П-101", operator: "Иванов И.И."}, {id: 2, desc: "Контроль давления", operator: "Петров П.П."}, {id: 3, desc: "Отбор проб", operator: "Иванов И.И."}

Вариант 2. Цепочка партий продукции.

Доп. поле: std::string batch_id; // Номер партии (например, "П-240501")

Специальная задача: Удалить из списка первую партию с заданным номером (номер задается константой). Вывести список до и после удаления.
Пример элементов: {id:1, desc:"Бензин АИ-92", batch:"П-240501"}, {id:2, desc:"Дизель Л-0,2", batch:"П-240502"}, {id:3, desc:"Мазут", batch:"П-240501"}.

Вариант 3. Очередь заявок на ремонт.

Доп. поле: int priority; // Приоритет (1-высокий, 2-средний, 3-низкий)
Специальная задача: Подсчитать количество заявок с высоким приоритетом (1).
Пример элементов: {id:1, desc:"Замена сальника насоса", priority:1}, {id:2, desc:"Чистка фильтра", priority:3}, {id:3, desc:"Ремонт задвижки", priority:1}.

Вариант 4. Маршрут движения пробы.

Доп. поле: std::string location; // Место отбора/анализа ("цех №1", "лаборатория", "склад")
Специальная задача: Вывести только те пробы, которые находятся в лаборатории.
Пример элементов: {id:1, desc:"Проба нефти сырой", location:"цех №1"}, {id:2, desc:"Проба бензина", location:"лаборатория"}.

Вариант 5. График поверки приборов КИП.

Доп. поле: std::string verification_date; // Дата очередной поверки (строка "ДД.ММ.ГГГГ")
Специальная задача: Найти прибор с самой ранней датой поверки (сравнение строк допустимо, если формат фиксирован).
Пример элементов: {id:1, desc:"Манометр МП-100", date:"15.10.2024"}, {id:2, desc:"Термопара ТХК", date:"10.09.2024"}.

Вариант 6. Журнал аварийных остановок.

Доп. поле: int downtime_minutes; // Время простоя, мин.
Специальная задача: Рассчитать суммарное время простоя по всем остановкам в списке.
Пример элементов: {id:1, desc:"Остановка из-за сбоя питания", downtime: 45}, {id:2, desc:"Остановка на плановый осмотр", downtime: 120}.

Вариант 7. Цепочка стадий процесса (например, очистки).

Доп. поле: std::string stage_type; // Тип стадии ("отстой", "фильтрация", "сушка")
Специальная задача: Вставить новую стадию "контроль качества" после каждой стадии типа "фильтрация". Вывести измененный список.
Пример элементов: {id:1, desc:"Первичный отстой", type:"отстой"}, {id:2, desc:"Песочный фильтр", type:"фильтрация"}.

Вариант 8. Список расходных материалов на складе.

Доп. поле: int quantity; // Количество на остатке
Специальная задача: Найти материал с минимальным остатком.
Пример элементов: {id:1, desc:"Прокладка сальниковая 10мм", quantity: 56}, {id:2, desc:"Фильтр воздушный ФВ-100", quantity: 2}.

Вариант 9. История изменений параметра установки.

Доп. поле: double parameter_value; // Значение параметра (температура, давление)
Специальная задача: Найти все значения параметра, превышающие заданный порог.
Пример элементов: {id:1, desc:"Температура в реакторе Р-1", value: 245.5}, {id:2, desc:"Температура в реакторе Р-1", value: 251.0}.

Вариант 10. Очередь на отгрузку продукции.

Доп. поле: std::string transport; // Вид транспорта ("автоцистерна", "ж/д цистерна", "трубопровод")
Специальная задача: Определить, есть ли в очереди отгрузка автоцистерной. Если да, вывести ее описание.
Пример элементов: {id:1, desc:"Отгрузка дизеля", transport:"ж/д цистерна"}, {id:2, desc:"Отгрузка бензина", transport:"автоцистерна"}.

Методика выполнения:

1. Определите структуру для данных.
2. Определите структуру узла списка: struct Node { TechEvent data; Node* next; };
3. Реализуйте функцию pushBack. Не забудьте обработать случай пустого списка (head == nullptr).
4. Реализуйте функцию printList (обход списка с помощью цикла while).
5. В main создайте список: Node* head = nullptr;
6. Добавьте элементы с помощью pushBack.
7. Реализуйте и примените функцию для специальной задачи.
8. Напишите функцию void clearList(Node*& head) для удаления всех узлов и освобождения памяти. Вызовите ее в конце.

Содержание отчета:

1. Титульный лист.
2. Цель работы.
3. Индивидуальный вариант.
4. Листинг программы (особенно структуры Node и ключевые функции работы со списком).
5. Результаты выполнения (скриншоты: исходный список, результат работы специальной функции).
6. Выводы.

Выводы:

1. В чем основное преимущество списка перед массивом (динамическим или статическим) для задач, подобной вашей?
2. Опишите алгоритм добавления элемента в конец однонаправленного списка. Почему для этого часто хранят не только голову (head), но и хвост (tail)?
3. Какая основная сложность возникла у вас при реализации операций со списком? Как работа с указателями влияет на надежность программы?

Контрольные вопросы:

1. Что такое "голова списка" (head)? Что она хранит в случае пустого списка?
2. Чем отличается удаление элемента из начала списка от удаления из середины или конца с точки зрения манипуляций указателями?
3. Какой объем памяти потребляет пустой список (один узел с указателем next)? Как это соотносится с пустым динамическим массивом?
4. В каких реальных производственных системах (SCADA, MES) могут использоваться структуры данных, подобные спискам? Приведите пример.

Лабораторная работа №4. Работа с двунаправленными списками

Цель: освоить принципы организации и алгоритмы работы с двунаправленными линейными списками, научиться: создавать, добавлять, удалять элементы, осуществлять поиск и обход списка в прямом и обратном направлениях, применить двунаправленный список для моделирования процессов, где требуется навигация в обе стороны (например, история изменений, перемещение по этапам).

Код и наименование индикатора достижения компетенции: ОПК-8.1 Составляет алгоритмы, пишет и отлаживает коды на языке программирования

Время выполнения: 2 часа.

Подготовка к работе (повторить):

- понятие узла (DNode) двунаправленного списка (данные + указатель на следующий + указатель на предыдущий элемент);
- базовые операции: инициализация, добавление в начало/конец, вставка после/перед заданным узлом, удаление узла, поиск, вывод списка в прямом и обратном порядке;

- особенности работы с указателями при манипуляциях с двунаправленным списком (корректное обновление next и prev);
- освобождение памяти всего списка.

Общее задание для всех вариантов:

Разработайте программу, которая:

- определяет структуру DNode (узел двунаправленного списка), содержащую: данные согласно вашему варианту (структуру TechChange или TechStage), указатель DNode* next на следующий узел, указатель DNode* prev на предыдущий узел;
- определяет функции для работы со списком: void pushBack(DNode*& head, DNode*& tail, ...) - добавление в конец (с обновлением tail); void printForward(const DNode* head) - вывод списка от головы к хвосту; void printBackward(const DNode* tail) - вывод списка от хвоста к голове;
- функцию для решения специальной задачи вашего варианта;
- в функции main() создает пустой список (инициализирует head = nullptr и tail = nullptr), добавляет в него не менее 5 элементов;
- выводит созданный список в прямом и обратном порядке (для проверки корректности связей);
- выполняет специальную задачу и выводит результат;
- корректно освобождает всю память, занятую списком.

Индивидуальные варианты заданий:

Базовые данные (хранятся в узле) для всех вариантов - структура:

- int change_id; // Уникальный номер изменения/этапа;
- std::string description; // Описание изменения или этапа.

Вариант 1. История изменений технологического регламента.

Доп. поле в данных: std::string author; // Автор изменения

Специальная задача: Найти последнее изменение, сделанное заданным автором (поиск начать с хвоста списка для эффективности).

Пример элементов:{id: 1, desc: "Увеличение температуры реакции", author: "Петров П.П."}, {id: 2, desc: "Корректировка давления", author: "Сидоров С.С."}, {id: 3, desc: "Уточнение времени цикла", author: "Петров П.П."}

Вариант 2. Маршрут перемещения контейнера с пробой.

Доп. поле: std::string checkpoint; // Контрольная точка ("цех А", "лаборатория", "архив")

Специальная задача: Удалить из списка все записи с контрольной точкой "лаборатория". Вывести список до и после.

Пример элементов: {id:1, desc:"Отбор пробы", checkpoint:"цех А"}, {id:2, desc:"Анализ", checkpoint:"лаборатория"}.

Вариант 3. Журнал смен операторов на пульте управления.

Доп. поле: std::string shift_time; // Время смены ("дневная", "ночная")

Специальная задача: Подсчитать количество ночных смен.

Пример элементов: {id:1, desc:"Смена Иванова И.И.", shift:"дневная"}, {id:2, desc:"Смена Петрова П.П.", shift:"ночная"}.

Вариант 4. График планово-предупредительных ремонтов (ППР).

Доп. поле: std::string equipment_code; // Код оборудования ("Н-101", "К-202")

Специальная задача: Вывести обратную хронологию (от последнего к первому) всех записей для оборудования с заданным кодом.

Пример элементов: {id:1, desc:"ППР насоса", code:"Н-101"}, {id:2, desc:"ППР колонны", code:"К-202"}, {id:3, desc:"Текущий ремонт Н-101", code:"Н-101"}.

Вариант 5. Цепочка согласований документа.

Доп. поле: std::string department; // Отдел ("ОТК", "ПЭО", "главный инженер")

Специальная задача: Найти и вывести запись, которая находится непосредственно перед согласованием в отделе "главный инженер" (использовать навигацию prev).

Пример элементов: {id:1, desc:"Согласование начальником цеха", department:"цех №1"}, {id:2, desc:"Согласование ОТК", department:"ОТК"}, {id:3, desc:"Согласование главным инженером", department:"главный инженер"}.

Вариант 6. Регистрация отклонений параметров.

Доп. поле: double deviation_value; // Величина отклонения от нормы
Специальная задача: Найти первое (самое раннее) отклонение, превышающее заданный порог по модулю (например, >5.0). Поиск вести от головы.
Пример элементов: {id:1, desc:"Отклонение давления", deviation: 2.1}, {id:2, desc:"Отклонение температуры", deviation: -5.5}.

Вариант 7. Протокол калибровки прибора.

Доп. поле: std::string calibration_result; // Результат ("годен", "брак", "требует настройки")
Специальная задача: Вставить новую запись "отправлен в ремонт" после каждой записи с результатом "брак". Вывести измененный список.
Пример элементов: {id:1, desc:"Калибровка манометра", result:"годен"}, {id:2, desc:"Калибровка расходомера", result:"брак"}.

Вариант 8. История модификаций ПО АСУ ТП.

Доп. поле: std::string version; // Версия ПО ("v1.2", "v1.3")
Специальная задача: Найти запись с максимальным номером версии (сравнение строк). Вывести ее и соседние записи (предыдущую и следующую, если они есть).
Пример элементов: {id:1, desc:"Добавление трендов", version:"v1.1"}, {id:2, desc:"Исправление ошибки связи", version:"v1.2"}.

Вариант 9. Журнал приема/передачи смены.

Доп. поле: std::string notes; // Замечания к смене ("все норма", "оборудование шумит")
Специальная задача: Вывести список в обратном порядке, но только те записи, где в замечаниях есть слово "норма".
Пример элементов: {id:1, desc:"Смена 06:00-14:00", notes:"все норма"}, {id:2, desc:"Смена 14:00-22:00", notes:"оборудование шумит"}.

Вариант 10. Трассировка сырья (от партии до продукта).

Доп. поле: std::string material_state; // Состояние материала ("сырье", "полуфабрикат", "готовый продукт")
Специальная задача: Определить, сколько этапов материал находился в состоянии "полуфабрикат".
Пример элементов: {id:1, desc:"Поступление нефти", state:"сырье"}, {id:2, desc:"После атмосферной перегонки", state:"полуфабрикат"}, {id:3, desc:"После гидроочистки", state:"готовый продукт"}.

Методика выполнения:

1. Определите структуру для данных (например, struct TechChange).
2. Определите структуру узла двунаправленного списка:
   cpp
   struct DNode {
      TechChange data;
      DNode* next;
      DNode* prev;
   };
3. Реализуйте функцию pushBack. Не забудьте обновлять как next нового узла, так и prev.
4. Обработайте случай пустого списка (обновление и head, и tail).
5. Реализуйте функции printForward и printBackward.
6. В main создайте список: DNode* head = nullptr; DNode* tail = nullptr;
7. Добавьте элементы с помощью pushBack.

8. Реализуйте и примените функцию для специальной задачи, используя преимущества двунаправленности (навигация от head или tail в зависимости от условия).
9. Напишите функцию void clearList(DNode*& head, DNode*& tail) для удаления всех узлов. Вызовите ее в конце.

Содержание отчета:

1. Титульный лист.
2. Цель работы.
3. Индивидуальный вариант.
4. Листинг программы (особенно структуры DNode и функции pushBack).
5. Результаты выполнения (скриншоты: вывод в прямом и обратном порядке, результат специальной задачи).
6. Выводы.

Выводы:

1. В чем принципиальное преимущество двунаправленного списка перед однонаправленным? В каких производственных задачах это преимущество критично?
2. Опишите алгоритм удаления элемента из середины двунаправленного списка. Сколько указателей нужно изменить?
3. Почему в двунаправленном списке часто хранят и голову (head), и хвост (tail)? Как это упрощает операции добавления в конец и обход в обратном порядке?

Контрольные вопросы:

1. Какие дополнительные расходы по памяти несет двунаправленный список по сравнению с однонаправленным? Оправданы ли они?
2. Что произойдет, если при удалении узла из середины списка забыть обновить один из указателей (next или prev) у соседних узлов?
3. Как можно реализовать циклический двунаправленный список? Какие новые возможности он дает?
4. Приведите пример из области АСУ ТП или ERP-систем, где для хранения данных логично использовать именно двунаправленный список, а не массив или однонаправленный список.

Лабораторная работа №5. Классы (инкапсуляция, наследование, полиморфизм)

Цель: освоить базовые принципы объектно-ориентированного программирования (ООП): инкапсуляцию, наследование, полиморфизм; научиться: проектировать и реализовывать иерархии классов для моделирования объектов предметной области (нефтехимическое производство); применить: механизм виртуальных функций для обеспечения полиморфного поведения.

Код и наименование индикатора достижения компетенции: ОПК-2.2 Применяет современные информационных технологий и программные средства, в том числе отечественного производства, при решении задач профессиональной деятельности; ОПК-8.1 Составляет алгоритмы, пишет и отлаживает коды на языке программирования

Время выполнения: 2 часа.

Подготовка к работе (повторить):

- понятие класса, объекта, инкапсуляции (спецификаторы доступа private, public, protected);
- конструкторы и деструкторы;
- наследование классов (ключевое слово class Derived : public Base);

- понятие виртуальной функции и полиморфизма. Чисто виртуальные функции и абстрактные классы.

Общее задание для всех вариантов:

Разработайте программу, которая:

- создает иерархию классов (базовый и производные) согласно вашему варианту;
- в базовом классе объявляет чисто виртуальную функцию (например, calculate() или getInfo()), делая класс абстрактным;
- в производных классах переопределяет (реализует) эту виртуальную функцию;
- в функции main() создает массив указателей на базовый класс, который заполняется адресами объектов производных классов;
- в цикле вызывает виртуальную функцию для каждого объекта, демонстрируя полиморфное поведение;
- программа должна демонстрировать работу конструкторов, деструкторов и корректно освобождать память.

Индивидуальные варианты заданий:

Базовый класс для всех вариантов: Equipment (Оборудование).

Общие защищенные (protected) поля:

- string name; // Наименование
- string id; // Инвентарный номер
- int year; // Год ввода в эксплуатацию

Публичные (public) методы:

- конструктор для инициализации полей;
- виртуальный метод void displayInfo() const; // Вывод общей информации;
- виртуальный метод virtual double calculateEfficiency() const = 0; // Расчет показателя эффективности (зависит от типа).

Вариант 1. Насосное оборудование.

Производный класс: Pump : public Equipment

Доп. поля: double flowRate; // Производительность, м³/ч; double power; // Мощность, кВт

Задача: Реализовать метод calculateEfficiency(), который возвращает КПД (условный) как flowRate / power. Метод displayInfo() выводит все данные, включая рассчитанный КПД.

Вариант 2. Теплообменное оборудование.

Производный класс: HeatExchanger : public Equipment

Доп. поля: double heatArea; // Площадь теплообмена, м²; double tempIn, tempOut; // Температуры на входе/выходе, °C

Задача: calculateEfficiency() возвращает разность температур tempIn - tempOut. displayInfo() выводит все данные.

Вариант 3. Резервуар (емкостное оборудование).

Производный класс: StorageTank : public Equipment

Доп. поля: double volume; // Объем, м³; string productType; // Тип хранимого продукта

Задача: calculateEfficiency() возвращает коэффициент использования (условный) как volume / 1000. displayInfo() выводит данные.

Вариант 4. Реактор.

Производный класс: Reactor : public Equipment

Доп. поля: double pressure; // Давление, атм; string catalyst; // Тип катализатора

Задача: calculateEfficiency() возвращает удельное давление (условный показатель) как pressure / 10. displayInfo() выводит данные.

Вариант 5. Колонна ректификационная.

Производный класс: DistillationColumn : public Equipment

Доп. поля: int traysNumber; // Количество тарелок; double height; // Высота, м

Задача: calculateEfficiency() возвращает "плотность тарелок" как traysNumber / height. displayInfo() выводит данные.

Вариант 6. Компрессор.

Производный класс: Compressor : public Equipment
Доп. поля: double compressionRatio; // Степень сжатия; string driveType; // Тип привода ("электрический", "турбинный")
Задача: calculateEfficiency() возвращает логарифм степени сжатия (условный) log(compressionRatio). displayInfo() выводит данные.

Вариант 7. Печь трубчатая.

Производный класс: Furnace : public Equipment
Доп. поля: double fuelConsumption; // Расход топлива, кг/ч; double heatOutput; // Тепловая мощность, Гкал/ч
Задача: calculateEfficiency() возвращает тепловой КПД как heatOutput / (fuelConsumption * 10) (условная формула). displayInfo() выводит данные.

Вариант 8. Сепаратор.

Производный класс: Separator : public Equipment
Доп. поля: double separationEfficiency; // Эффективность сепарации, %; string principle; // Принцип действия ("гравитационный", "центробежный")
Задача: calculateEfficiency() возвращает separationEfficiency. displayInfo() выводит данные.

Вариант 9. Насос-дозатор (химический).

Производный класс: DosingPump : public Equipment
Доп. поля: double dosingAccuracy; // Точность дозирования, %; string reagent; // Название реагента
Задача: calculateEfficiency() возвращает 100 - dosingAccuracy (чем меньше погрешность, тем выше эффективность). displayInfo() выводит данные.

Вариант 10. Фильтр.

Производный класс: Filter : public Equipment
Доп. поля: double poreSize; // Размер пор, мкм; string filteredMedia; // Фильтруемая среда
Задача: calculateEfficiency() возвращает условную "тонкость фильтрации" как 1 / poreSize. displayInfo() выводит данные.

Методика выполнения:

1. Объявите базовый абстрактный класс Equipment с чисто виртуальной функцией calculateEfficiency().
2. Объявите производный класс согласно вашему варианту.
3. В производном классе реализуйте конструктор, который инициализирует и свои поля, и поля базового класса (список инициализации).
4. Переопределите метод displayInfo() для вывода всех полей (базовых и производных).
5. Реализуйте метод calculateEfficiency() согласно формуле вашего варианта.
6. В main() создайте массив указателей на Equipment* equipArray[3];.
7. Заполните его адресами динамически созданных объектов ваших производных классов (можно 2-3 объекта одного типа или разных, если позволяет вариант).
    - cpp
    - equipArray[0] = new Pump("Насос Н-1", "INV001", 2020, 500.0, 75.0);
8. В цикле вызовите equipArray[i]->displayInfo() и cout << "Эффективность: " << equipArray[i]->calculateEfficiency() << endl;.
9. Не забудьте в конце цикла освободить память: delete equipArray[i];.

Содержание отчета:

1. Титульный лист.
2. Цель работы.

3. Индивидуальный вариант.
4. Листинг программы (полные объявления классов, особенно важно показать иерархию и виртуальные функции).
5. Результаты выполнения (скриншот вывода программы).
6. Выводы

Выводы:

1. Объясните, в чем проявилась инкапсуляция в вашей программе (на примере полей класса).
2. Как работает механизм полиморфизма? Что произошло бы, если бы функция calculateEfficiency() не была объявлена виртуальной?
3. Какие практические преимущества дает использование иерархии классов и полиморфизма при разработке ПО для автоматизации производства? Приведите пример из вашей предметной области.

Контрольные вопросы:

1. Можно ли создать объект абстрактного класса Equipment? Почему?
2. В какой момент (во время компиляции или выполнения) определяется, какая реализация виртуальной функции будет вызвана через указатель на базовый класс? Как называется этот механизм?
3. Что такое таблица виртуальных функций (vtable) и зачем она нужна?
4. Предложите, как можно расширить иерархию для вашего варианта. Например, от класса Pump унаследовать CentrifugalPump и PistonPump. Какие новые поля и методы они могли бы иметь?

Лабораторная работа №6. Работа с операциями (перегрузка операторов)

Цель: освоить синтаксис и принципы перегрузки операторов в C++. Научиться: перегружать основные операторы (арифметические, сравнения, ввода/вывода) для пользовательских типов данных, моделирующих объекты нефтехимического производства; применить: перегрузку для интуитивно понятной работы с объектами.

Код и наименование индикатора достижения компетенции: ОПК-6.1 Выявляет потребности организации в компьютерном и сетевом оборудовании

Время выполнения: 2 часа.

Подготовка к работе (повторить):

- понятие перегрузки операторов. Операторы, которые можно и нельзя перегружать;
- способы перегрузки: как член класса (метод) и как дружественная функция (friend);
- перегрузка операторов ввода (>>) и вывода (<<);
- перегрузка арифметических операторов (+, -, *, /) и операторов сравнения (==, !=, <, >).

Общее задание для всех вариантов:

Разработайте программу, которая:

- определяет класс согласно вашему варианту (например, Chemical, Apparatus, Stream);
- в классе перегружает необходимый набор операторов (см. вариант);
- в функции main() демонстрирует работу всех перегруженных операторов на созданных объектах;
- программа должна быть логичной и демонстрировать осмысленное применение перегрузки в предметной области.

Индивидуальные варианты заданий:

Вариант 1. Класс «Химический реагент» (ChemicalReagent).

Поля: string name; (название), double concentration; (концентрация, %), double volume; (объем, л).
Перегрузить операторы:
+ (сложение двух реагентов): возвращает новый реагент с именем "Смесь", концентрацией как средневзвешенная по объему, объемом как сумма объемов.
== (сравнение): равны, если совпадают имя И концентрация с точностью до 0.1%.
<< (вывод): выводит данные в формате "Reagent: [name], Conc: [concentration]%, Vol: [volume] L".
Демонстрация: Создать 2-3 реагента, сложить их, сравнить, вывести.

Вариант 2. Класс «Поток вещества» (MaterialStream).

Поля: string composition; (состав, например, "Нефть"), double massFlow; (массовый расход, кг/ч), double temperature; (температура, °C).
Перегрузить операторы:
+ (объединение потоков): если состав одинаковый, складываются расходы, температура считается как средневзвешенная по массе. Если состав разный, имя становится "Смешанный поток".
> (сравнение): сравнение по массовому расходу.
<< (вывод): выводит данные в удобном формате.
Демонстрация: Создать потоки, объединить, сравнить по расходу.

Вариант 3. Класс «Точка контроля» (MonitoringPoint).

Поля: int id;, double pressure; (давление, атм), double temperature; (температура, °C).
Перегрузить операторы:
- (разность двух точек): возвращает новую точку, где поля – разности соответствующих параметров (для анализа перепада).
== (сравнение): точки равны, если оба параметра совпадают с допуском.
>> (ввод): позволяет ввести данные точки с клавиатуры.
Демонстрация: Ввести 2 точки, вычислить перепад, сравнить.

Вариант 4. Класс «Партия продукции» (ProductBatch).

Поля: string productCode; (код продукта), double quantity; (количество, т), double qualityIndex; (показатель качества, баллы).
Перегрузить операторы:
* (умножение на коэффициент): имитирует отбор пробы (уменьшение количества) или масштабирование.
< (сравнение): сравнение по показателю качества.
<< и >> для ввода/вывода.
Демонстрация: Создать партию, "взять пробу" (умножить на 0.01), сравнить с другой партией по качеству.

Вариант 5. Класс «Аппарат» (Apparatus).

Поля: string name;, double volume; (объем, м³), double occupancy; (заполненность, % от 0 до 100).
Перегрузить операторы:
+= (добавление загрузки): увеличивает заполненность на заданный процент с проверкой на переполнение (>100%).
-= (разгрузка): уменьшает заполненность.
== (сравнение): равны, если объем и имя совпадают.
Демонстрация: Загрузить и разгрузить аппарат, проверить на переполнение, сравнить аппараты.

Вариант 6. Класс «Интервал времени» (TimeInterval) для ППР.

Поля: string startDate; (дата начала, строка), int durationDays; (длительность, дни).
Перегрузить операторы:
+ (сложение с числом): сдвиг интервала вперед по времени (увеличивает startDate условно, для простоты можно увеличивать durationDays).

> (сравнение): какой интервал позже? Сравнение по дате начала.
<< (вывод).
Демонстрация: Создать интервал планового ремонта, сдвинуть его, сравнить с другим.

Вариант 7. Класс «Катализаторная загрузка» (CatalystCharge).

Поля: string type;, double mass; (масса, кг), int activity; (активность, усл. ед.).
Перегрузить операторы:
/ (деление на целое число): имитирует деление загрузки на несколько реакторов (уменьшение массы, активность та же).
!= (неравенство): сравнивает по типу и активности.
<< (вывод).
Демонстрация: Разделить загрузку, сравнить разные загрузки.

Вариант 8. Класс «Энергопотребление» (EnergyConsumption).

Поля: string unitName; (наименование агрегата), double power; (мощность, кВт), double hours; (время работы, ч).
Перегрузить операторы:
* (расчет потребленной энергии): возвращает power * hours (кВт*ч).
Можно реализовать как оператор умножения двух объектов (некорректно семантически) или как оператор * между объектом и числом (время). Лучше перегрузить оператор приведения к double() или отдельный метод.
Уточнение: Перегрузите оператор * между объектом и числом (время), который возвращает новое значение потребленной энергии.
+= (добавление времени работы).
<< (вывод).
Демонстрация: Рассчитать потребление за разное время.

Вариант 9. Класс «Координата» (Coordinate) для размещения оборудования.

Поля: double x, y, z; (координаты в метрах).
Перегрузить операторы:
- (вычисление расстояния между двумя координатами).
== (совпадение координат).
>> и << для ввода/вывода.
Демонстрация: Ввести координаты двух аппаратов, вычислить расстояние между ними.

Вариант 10. Класс «Датчик» (Sensor).

Поля: string parameter; ("давление", "температура"), double value; (текущее значение), double error; (погрешность, %).
Перегрузить операторы:
++ (префиксный): имитирует считывание нового, слегка зашумленного значения (добавить небольшой случайный шум в пределах погрешности). Для простоты можно просто увеличить value на 0.1.
> (сравнение): сравнение по значению с учетом погрешности? Или просто по значению.
<< (вывод).
Демонстрация: "Считать" показания датчика несколько раз (использовать ++), сравнить показания.

Методика выполнения:

1. Объявите класс с приватными полями и публичными методами доступа (геттеры/сеттеры), если это необходимо для инкапсуляции.
2. Реализуйте перегрузку операторов указанными способами. Операторы ввода/вывода (<<, >>) перегружаются как дружественные функции. Бинарные операторы (например, +, -, ==), где левый операнд – объект вашего класса, можно

перегружать как методы класса или как дружественные функции. Операторы, изменяющие объект (+=, -=, ++), лучше перегружать как методы класса.
3. В main() создайте 2-3 объекта класса.
4. Продемонстрируйте использование каждого перегруженного оператора, выводя результаты на экран.
5. Продумайте обработку граничных случаев (например, переполнение аппарата, деление на ноль).

Содержание отчета:

1. Титульный лист.
2. Цель работы.
3. Индивидуальный вариант.
4. Листинг программы (особенно объявления и реализации перегруженных операторов).
5. Результаты выполнения (скриншоты с демонстрацией всех операций).
6. Выводы.

Выводы:

1. Какие преимущества дает перегрузка операторов для вашего класса с точки зрения читаемости и удобства кода?
2. В чем разница между перегрузкой оператора как метода класса и как дружественной функции? Когда какой способ предпочтительнее (приведите пример из вашего кода)?
3. Может ли перегрузка операторов ввести в заблуждение пользователя класса? Как этого избежать (принцип "не удивляй пользователя")?

Контрольные вопросы:

1. Почему оператор присваивания (=) и оператор индексирования ([]) часто перегружают как методы класса, а не как дружественные функции?
2. Можно ли перегрузить оператор . (точка) или :: (область видимости)? Почему?
3. Что такое "префиксная" и "постфиксная" форма операторов инкремента/декремента? Как их отличить при перегрузке?
4. Приведите пример из химической технологии, где перегрузка оператора сравнения (<, >) для двух объектов имела бы ясный физический смысл.

Лабораторная работа №7. Наследование и виртуальные функции

Цель: углубить понимание механизмов наследования и полиморфизма в C++; научиться: строить сложные иерархии классов, использовать абстрактные классы, виртуальные деструкторы, чисто виртуальные функции; применить: полиморфизм для обработки разнородных объектов в едином контейнере в контексте системы управления производством.

Код и наименование индикатора достижения компетенции: ОПК-8.1 Составляет алгоритмы, пишет и отлаживает коды на языке программирования, ОПК-8.3 Тестирует работоспособность программы.

Время выполнения: 2 часа.

Подготовка к работе (повторить):

- множественное наследование и виртуальное наследование;
- виртуальные функции и абстрактные классы;
- виртуальный деструктор (когда и зачем он нужен);
- ключевое слово override (C++11 и выше);
- приведение типов в иерархиях: dynamic_cast, typeid.

Общее задание для всех вариантов:

Разработайте программу, которая:

- создает иерархию классов с минимум тремя уровнями (например, Базовый -> Промежуточный -> Конкретный);
- в корне иерархии создает абстрактный класс с чисто виртуальными функциями;
- использует механизм виртуального деструктора;
- использует ключевое слово override для явного указания переопределения;
- в функции main() создает единый контейнер (например, vector<указатель_на_базовый_класс>), содержащий объекты разных конкретных классов из иерархии;
- демонстрирует полиморфное поведение через вызов виртуальных функций для всех объектов в контейнере.

Задание повышенной сложности (для всех): реализуйте функцию testIntegrity(), которая "тестирует" объект (проверяет его внутреннюю согласованность или выводит специнформацию) и вызывайте ее для каждого объекта. Это моделирует аспект тестирования (ОПК-8.3).

Индивидуальные варианты заданий:

Общая концепция: моделирование элементов системы управления технологическим процессом (АСУ ТП).

Вариант 1. Иерархия «Технологический параметр».

Уровень 1 (абстрактный): TechParameter (чисто виртуальные: getCurrentValue(), getUnit(), testIntegrity()).
Уровень 2 (базовые типы): AnalogParameter (доп. поле double minRange, maxRange), DiscreteParameter (доп. поле int states).
Уровень 3 (конкретные):
- TemperatureSensor : public AnalogParameter (реализует методы, testIntegrity() проверяет, в пределах ли диапазона значение).
- PressureSensor : public AnalogParameter (аналогично).
- ValvePosition : public DiscreteParameter (states=2 "открыт/закрыт", testIntegrity() всегда возвращает true).
Задача: Создать вектор указателей на TechParameter, добавить объекты всех типов. В цикле вывести значения, единицы измерения и результаты "теста".

Вариант 2. Иерархия «Устройство КИПиА».

Уровень 1: Instrument (чисто виртуальные: getType(), performSelfTest(), calibrate()).
Уровень 2: FieldDevice (поле string location), ControlRoomDevice.
Уровень 3:
- FlowTransmitter : public FieldDevice (реализует методы).
- PLC : public ControlRoomDevice (реализует методы, performSelfTest() может возвращать случайный статус).
- ChartRecorder : public ControlRoomDevice.
Задача: Вектор указателей на Instrument. Выполнить для всех performSelfTest() и вывести результаты.

Вариант 3. Иерархия «Событие в системе».

Уровень 1: SystemEvent (чисто виртуальные: getSeverity(), getDescription(), acknowledge()).
Уровень 2: AlarmEvent (поле int priority), LogEvent.
Уровень 3:
- HighPriorityAlarm : public AlarmEvent (priority=1).
- LowPriorityAlarm : public AlarmEvent (priority=3).
- OperatorActionLog : public LogEvent (acknowledge() ничего не делает).
Задача: Создать список событий, вывести описание и квитировать (acknowledge) все аварии.

Вариант 4. Иерархия «Объект ремонта».

Уровень 1: MaintainableItem (чисто виртуальные: getMaintenanceInterval(), performMaintenance(), isOverdue()).
Уровень 2: RotatingEquipment (поле int rpm), StaticEquipment.
Уровень 3:
- Pump : public RotatingEquipment.
- HeatExchanger : public StaticEquipment.
- SafetyValve : public StaticEquipment (интервал обслуживания всегда 365 дней).

Задача: Вектор указателей. Для каждого вывести интервал и проверить, не просрочено ли ТО (isOverdue).

Вариант 5. Иерархия «Документация».

Уровень 1: Document (чисто виртуальные: getDocNumber(), print(), verify()).
Уровень 2: RegulatoryDoc (поле string standard), OperationalDoc.
Уровень 3:
- PipingDiagram : public OperationalDoc.
- SafetyInstruction : public RegulatoryDoc.
- TechManual : public OperationalDoc.

Задача: Создать массив документов, вызвать для каждого print() и verify().

Вариант 6. Иерархия «Режим работы установки».

Уровень 1: OperationMode (чисто виртуальные: getName(), calculateEfficiency(), validateParameters()).
Уровень 2: SteadyStateMode, TransitionMode.
Уровень 3:
- FullProductionMode : public SteadyStateMode (высокий КПД).
- StartupMode : public TransitionMode (низкий КПД).
- ShutdownMode : public TransitionMode.

Задача: Создать список режимов, рассчитать и сравнить их эффективность.

Вариант 7. Иерархия «Химический компонент потока».

Уровень 1: StreamComponent (чисто виртуальные: getFormula(), getConcentration(), isHazardous()).
Уровень 2: Hydrocarbon, InorganicCompound.
Уровень 3:
- Methane : public Hydrocarbon.
- Water : public InorganicCompound.
- HydrogenSulfide : public InorganicCompound (isHazardous() возвращает true).

Задача: Вектор компонентов. Вывести формулу и проверить, является ли опасным.

Вариант 8. Иерархия «Блок управления».

Уровень 1: ControlUnit (чисто виртуальные: getState(), sendCommand(string), diagnose()).
Уровень 2: AnalogController, DigitalController.
Уровень 3:
- PIDController : public AnalogController.
- OnOffController : public DigitalController.
- ProgrammableController : public DigitalController.

Задача: Создать массив блоков, отправить каждому тестовую команду, запустить диагностику (diagnose).

Вариант 9. Иерархия «Средство защиты».

Уровень 1: SafetyDevice (чисто виртуальные: getProtectionLevel(), activate(), test()).
Уровень 2: PassiveProtection, ActiveProtection.
Уровень 3:
- FireWall : public PassiveProtection.
- EmergencyShutdownValve : public ActiveProtection.

- GasDetector : public ActiveProtection (test() имитирует проверку чувствительности).
Задача: Протестировать (test()) все устройства защиты в системе.

Вариант 10. Иерархия «Запись в базу данных Historian».

Уровень 1: HistoryRecord (чисто виртуальные: getTimestamp(), getValueAsString(), isAnomaly()).
Уровень 2: AnalogRecord, StringRecord.
Уровень 3:
- PressureRecord : public AnalogRecord (isAnomaly() срабатывает при значении > 100).
- StatusRecord : public StringRecord (значение "ON"/"OFF").
- AlarmRecord : public StringRecord (всегда аномалия).

Задача: Создать архив записей, вывести их и пометить аномальные.

Методика выполнения:

1. Продумайте иерархию, определите поля и методы для каждого уровня.
2. Абстрактный класс должен иметь хотя бы одну чисто виртуальную функцию и виртуальный деструктор (virtual ~ClassName() {} или = default).
3. В производных классах используйте override для ясности.
4. Реализуйте функцию testIntegrity() или ее аналог из вашего варианта.
5. В main():

```
cpp
std::vector<BaseClass*> system;
system.push_back(new DerivedClass1(...));
system.push_back(new DerivedClass2(...));
// ...
for (auto* obj : system) {
    obj->someVirtualFunction();
    obj->testIntegrity(); // Демонстрация тестирования
}
// Очистка
for (auto* obj : system) delete obj;
```

Содержание отчета:

1. Титульный лист.
2. Цель работы.
3. Индивидуальный вариант (схему иерархии можно изобразить графически или текстом).
4. Листинг программы (полная иерархия классов, особое внимание на виртуальные функции, деструкторы, override).
5. Результаты выполнения (скриншот вывода, демонстрирующий полиморфизм и работу testIntegrity).
6. Выводы.

Выводы:

1. Объясните, почему в данной работе виртуальный деструктор является обязательным? Что произошло бы без него?
2. Как использование ключевого слова override помогает предотвратить ошибки при разработке и модификации иерархии классов?
3. Как продемонстрированный подход (единый контейнер указателей на базовый класс) может быть использован в реальной SCADA-системе или MES для управления разнородными объектами? Свяжите с компетенцией ОПК-8.3 (тестирование).

Контрольные вопросы:

1. В чем разница между виртуальной и чисто виртуальной функцией? Можно ли иметь чистую виртуальную функцию с телом в C++?

2. Что такое «срезка объекта» (object slicing) и при каких операциях она может произойти? Как ее избежать в контексте данной работы?
3. Для чего используется оператор dynamic_cast? Приведите пример его осмысленного использования в вашей иерархии классов.
4. Как механизм позднего связывания (полиморфизм) влияет на производительность программы? Всегда ли его использование оправдано?

Лабораторная работа №8. Наследование и виртуальные функции

Цель: освоить практическое применение контейнеров, алгоритмов и итераторов Standard Template Library (STL) для решения типовых задач обработки данных; научиться: выбирать подходящий контейнер (vector, list, map, set), применять к нему стандартные алгоритмы (sort, find, copy, for_each), использовать лямбда-выражения; применить STL для анализа и обработки производственных данных.

Код и наименование индикатора достижения компетенции: ОПК-2.2 Применяет современные информационных технологий и программные средства, в том числе отечественного производства, при решении задач профессиональной деятельности; ОПК-8.1 Составляет алгоритмы, пишет и отлаживает коды на языке программирования.

Время выполнения: 2 часа.

Подготовка к работе (повторить):

- основные контейнеры STL: последовательные (vector, list, deque), ассоциативные (map, set), адаптеры (stack, queue);
- понятие итератора, категории итераторов;
- алгоритмы STL (<algorithm>): немодифицирующие (find, count), модифицирующие (copy, transform), сортировки и бинарный поиск (sort, binary_search);
- лямбда-выражения (с [capture](params) -> ret { body });
- функциональные объекты и связыватели (bind).

Общее задание для всех вариантов:

Разработайте программу, которая:

- определяет структуру или класс для хранения единицы данных согласно вашему варианту;
- использует минимум два разных контейнера STL (например, vector и map, или set и list);
- применяет минимум три разных алгоритма STL (например, sort, find_if, for_each, transform);
- использует лямбда-выражения хотя бы в одном алгоритме;
- заполняет контейнеры тестовыми данными (не менее 7-10 записей);
- выполняет задачу вашего варианта, эффективно используя STL, и выводит результаты.

Индивидуальные варианты заданий:

Базовый тип данных для всех вариантов: ProductionRecord (Запись о производстве).

Общие поля: int id;, string date; (дата), string productCode; (код продукта), double quantity; (количество, т).

Вариант 1. Анализ суточного выпуска продукции.

Контейнеры: vector<ProductionRecord>, map<string, double> (код продукта -> суммарный выпуск).

Алгоритмы и задача:

sort вектора по дате.

for_each с лямбдой для заполнения map (суммирование количества по коду продукта).

find_if в векторе для поиска первой записи, где количество > X (заданное значение).
max_element в map для поиска продукта с максимальным суммарным выпуском.
Вывод: Отсортированный журнал, суммарный выпуск по продуктам, результат поиска.

Вариант 2. Контроль качества партий.

Доп. поле в данных: double qualityScore; (оценка качества, 0-100).
Контейнеры: list<ProductionRecord>, set<string> (коды продуктов, прошедших контроль).
Алгоритмы и задача:
remove_if с лямбдой для удаления из списка записей с оценкой < 85.
copy_if оставшихся записей во временный вектор.
transform для извлечения кодов продуктов из этого вектора и вставки в set (уникальные коды).
count_if для подсчета партий с оценкой > 95.
Вывод: Список после очистки, уникальные коды качественных продуктов, количество отличных партий.

Вариант 3. Поиск аномалий в данных.

Доп. поле: double expectedQuantity; (плановое количество).
Контейнеры: deque<ProductionRecord>, vector<ProductionRecord> для аномалий.
Алгоритмы и задача:
generate_n для заполнения deque тестовыми данными.
copy_if с лямбдой в новый вектор записей, где отклонение abs(quantity - expectedQuantity) > 0.1 * expectedQuantity.
sort вектора аномалий по величине отклонения (по убыванию).
accumulate для подсчета суммарного отклонения по всем аномалиям.
Вывод: Исходные данные, список аномалий, суммарное отклонение.

Вариант 4. Статистика по сменам.

Доп. поле: string shift; ("день", "ночь").
Контейнеры: vector<ProductionRecord>, multimap<string, ProductionRecord> (смена -> запись).
Алгоритмы и задача:
Заполнить multimap из вектора.
equal_range для получения диапазона записей по смене "ночь".
for_each с лямбдой для этого диапазона для подсчета общего количества за ночь.
partition вектора на записи дневной и ночной смен.
Вывод: Записи ночной смены, общий объем ночного выпуска, вектор после разделения.

Вариант 5. Ранжирование оборудования по загрузке.

Доп. поле: string unitId; (код установки/аппарата).
Контейнеры: list<ProductionRecord>, map<string, double> (оборудование -> суммарная загрузка).
Алгоритмы и задача:
for_each с лямбдой для заполнения map.
Скопировать map в vector<pair<string,double>>.
sort этого вектора по убыванию загрузки (второй элемент пары).
find в map загрузку для конкретного аппарата (заданного кодом).
Вывод: Рейтинг оборудования по загрузке, загрузка заданного аппарата.

Вариант 6. Управление складскими остатками (FIFO).

Контейнеры: queue<ProductionRecord> (очередь поступления), vector<double> (для выборок).
Алгоритмы и задача:
Имитировать поступление партий, добавляя их в очередь.

Имитировать отгрузку: извлечь N первых партий (суммарно близко к требуемому объему отгрузки), записывая их id в вектор.
accumulate для подсчета отгруженного объема.
find в векторе id конкретной партии (была ли отгружена?).
Вывод: Состояние очереди до и после отгрузки, отгруженный объем.

Вариант 7. Анализ периодичности операций.

Доп. поле: string operationType; (тип операции: "загрузка", "реакция", "разгрузка").
Контейнеры: vector<ProductionRecord>, set<string> (уникальные операции).
Алгоритмы и задача:
sort вектора по operationType, затем по date.
unique_copy в set для получения уникальных типов операций.
adjacent_find с лямбдой для поиска двух подряд идущих записей с одинаковой операцией (возможная ошибка?).
count для подсчета операций определенного типа.
Вывод: Упорядоченный журнал операций, уникальные типы, результат поиска дубликатов.

Вариант 8. Калькулятор себестоимости.

Доп. поля: double rawMaterialCost;, double energyCost;.
Контейнеры: vector<ProductionRecord>, stack<double> (для промежуточных расчетов).
Алгоритмы и задача:
transform с лямбдой для вычисления общей себестоимости (rawMaterialCost + energyCost) и записи в новое поле или отдельный вектор.
sort по себестоимости (возрастание).
for_each для помещения себестоимости каждой партии в стек.
Вычислить среднюю себестоимость, извлекая значения из стека.
Вывод: Партии, отсортированные по себестоимости, средняя себестоимость.

Вариант 9. Фильтрация и группировка данных.

Контейнеры: vector<ProductionRecord>, map<string, vector<ProductionRecord>> (продукт -> вектор его записей).
Алгоритмы и задача:
stable_partition разделить вектор на записи за 2024 и 2025 годы (по дате).
Для каждой записи добавлять ее в соответствующий вектор внутри map.
Для каждого продукта в map применить minmax_element по полю quantity для нахождения минимальной и максимальной партии.
all_of проверить, все ли записи для заданного продукта имеют количество > 0.
Вывод: Группировка по продуктам, min/max для каждого, результат проверки.

Вариант 10. Построение рейтинга продукции.

Доп. поле: int customerRating; (рейтинг от покупателя, 1-5).
Контейнеры: list<ProductionRecord>, priority_queue<pair<double, string>> (приоритет = средний рейтинг).
Алгоритмы и задача:
Рассчитать средний рейтинг для каждого продукта (используйте дополнительную map для суммирования рейтингов и подсчета).
Поместить пары (средний рейтинг, код продукта) в priority_queue.
remove из списка все записи с рейтингом 1.
reverse список, чтобы увидеть последние записи первыми.
Вывод: Рейтинг продуктов (от высшего), очищенный и перевернутый список записей.

Методика выполнения:

1. Определите структуру ProductionRecord с полями вашего варианта.
2. Включите необходимые заголовки: <vector>, <map>, <algorithm>, <numeric>, <iterator>.
3. Подготовьте набор тестовых данных (можно инициализировать в коде).

4. Последовательно реализуйте пункты задачи, активно используя синтаксис STL и лямбда-выражения.

```
cpp
// Пример лямбды для sort
sort(v.begin(), v.end(), [](const ProductionRecord& a, const ProductionRecord& b) {
    return a.quantity > b.quantity;
});
// Пример for_each с лямбдой для заполнения map
for_each(v.begin(), v.end(), [&sumMap](const ProductionRecord& rec) {
    sumMap[rec.productCode] += rec.quantity;
});
```

5. Выводите промежуточные и окончательные результаты.

Содержание отчета:

1. Титульный лист.
2. Цель работы.
3. Индивидуальный вариант.
4. Листинг программы (особенно важны моменты использования контейнеров, алгоритмов и лямбда-выражений).
5. Результаты выполнения (скриншоты с выводами).
6. Выводы.

Выводы:

1. Почему вы выбрали именно эти контейнеры STL для своей задачи? Обоснуйте выбор с точки зрения эффективности операций, которые вам были нужны (вставка, поиск, удаление).
2. В чем преимущество использования алгоритмов STL (sort, find_if) по сравнению с написанием собственных циклов? Сравните по критериям читаемости кода, надежности и потенциальной производительности.
3. Как использование STL способствует достижению компетенции ОПК-2.2 (применение современных программных средств)? Сформулируйте, как знание STL повышает профессиональную ценность разработчика в области автоматизации производств.

Контрольные вопросы:

1. В чем ключевая разница между map и unordered_map? В каком случае предпочтительнее каждый из них в контексте вашей задачи?
2. Что такое «инвалидация итераторов»? Приведите пример операции с контейнером STL, которая приводит к инвалидации итераторов.
3. Как работает алгоритм sort для пользовательского типа данных? Что нужно, чтобы отсортировать вектор ваших структур по произвольному полю?
4. Какие ограничения накладывает использование алгоритмов STL (например, sort) на элементы контейнера (требования к итераторам, к самим элементам)?

7. Критерии и показатели оценки результата выполнения лабораторных работ

Оценка обучающемуся за выполненную лабораторную работу выставляется по четырехбалльной шкале по итогам проверки отчета и защиты работы. Основными **критериями оценки** лабораторной работы по дисциплине «Технологии программирования», являются:

- полнота и правильность выполнения задания (соответствие программы поставленному заданию, обработка всех требуемых случаев);
- качество кода (читаемость, структурированность, наличие комментариев, использование современных возможностей языка);

- оформление отчета (соответствие шаблону, наличие всех разделов, аккуратность, грамотность выводов);
- ответы на вопросы на защите отчета (понимание теоретических основ и практических аспектов выполненной работы);

В таблице 4 представлены критерии и показатели оценки выполнения лабораторной работы

Таблица 4

Критерии и показатели оценки выполнения лабораторной работы

Оценка	**Критерии**			
	Полнота и правильность выполнения	**Качество кода**	**Оформление отчета**	**Ответы на контрольные вопросы/защита**
«Отлично»	Задание выполнено полностью, код корректен.	Код чистый, хорошо структурирован, адекватно прокомментирован.	Отчет оформлен в соответствии с требованиями, выводы обоснованные.	Полные, уверенные, правильные ответы.
«Хорошо»	Задание выполнено с 1-2 незначительными недочетами.	Код читаем, но есть небольшие нарушения стиля.	Отчет оформлен с 1-2 замечаниями, выводы есть.	Ответы в целом правильные, но с неточностями.
«Удовлетвори-тельно»	Задание выполнено частично, есть ошибки в логике.	Код плохо отформатирован, минимальные комментарии.	Отчет оформлен небрежно, выводы поверхностные.	Ответы неполные, требуются наводящие вопросы.
«Неудовлетвори-тельно»	Задание не выполнено или выполнено неверно.	Код нечитаем.	Отчет не оформлен или оформление не соответствует требованиям.	Не может ответить на вопросы по теме.

Некоторые показатели, которые могут использоваться для оценки результата:

- функциональность – работающая реализация всех заявленных функций и возможностей.
- тестирование – наличие тестов и их результаты (покрытие кода тестами).
- креативность – оригинальные решения и подходы к реализации задачи.
- обработка ошибок – корректная обработка исключительных ситуаций и ошибок.
- пользовательский интерфейс – удобство и интуитивность интерфейса (если применимо).
- соответствие стандартам кодирования – следование принятым стандартам и стилям кодирования.

В качестве дополнительных критериев оценки лабораторной работы может быть: скорость выполнения программы, использование памяти и ресурсов, оптимизация алгоритмов и структур данных, выполнение требования к интерфейсу или взаимодействию с пользователем, используемые библиотеки и технологии, также учитывается, предоставлено ли описание работы программы, добавлены ли инструкции по запуску и использованию, документированы функции и классы в коде.

Приложение А
Образец титульного листа отчета по лабораторной работе

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования «Самарский государственный технический университет»
(ФГБОУ ВО «СамГТУ»)

Филиал ФГБОУ ВО «СамГТУ» в г. Новокуйбышевске

Кафедра «Информатика и системы управления»

ОТЧЕТ
по лабораторной работе № ____
по дисциплине «Технологии программирования»
Тема: _____

Выполнил:
обучающийся ___ курса _____ группы
ФИО: _____

Принял:
ФИО: _____

Оценка: _____

Дата: _____

Новокуйбышевск, 20___

Приложение Б

Образец отчета по лабораторной работе

---

1. Цель работы:

Сформировать навык использования структур для агрегации разнотипных данных, моделирующих сущность предметной области (нефтехимическое производство).

Научиться создавать структуры, инициализировать их, передавать в функции и производить над ними операции.

2. Подготовка к работе (Повторение)

Перед выполнением работы был повторен следующий теоретический материал:

- понятие структуры (*struct*) в языке C/C++ как типа данных, объединяющего разнородные данные (поля);
- способы инициализации структур (списочная инициализация, через конструктор);
- доступ к полям структуры с помощью оператора точки;
- передача структур в функции по значению (копия) и по ссылке (*&*) для избежания копирования и возможности изменения.

3. Ход работы (Программа выполнения)

3.1. Изучено задание, выбран вариант №1 (Насосное оборудование).

3.2. Создан новый проект в среде разработки.

3.3. Объявлена структура *Equipment* с базовыми полями и дополнительным полем *flow_rate*.

3.4. В функции *main()* создан массив (вектор) структур *Equipment* и заполнен данными согласно варианту.

3.5. Реализована функция *print_equipment()* для вывода информации об одном объекте.

3.6. Реализована функция *find_max_flow_rate_pump()* для поиска насоса с максимальной производительностью.

3.7. Реализован вывод исходного списка и результата работы специальной функции.

4. Программный код

cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <limits> // Для numeric_limits

using namespace std;

// 1. Определение структуры Equipment
struct Equipment {
    int id;                 // Уникальный идентификатор
    string name;            // Наименование оборудования
    string workshop;        // Цех/установка
    int year_commissioned;  // Год ввода в эксплуатацию
    double flow_rate;       // Производительность, м³/ч (для варианта 1)
};

// 2. Функция для печати одного элемента оборудования
void print_equipment(const Equipment& item) {
```

```
    cout << "ID: " << item.id
         << " | Наименование: " << item.name
         << " | Цех: " << item.workshop
         << " | Год ввода: " << item.year_commissioned
         << " | Производительность: " << item.flow_rate << " м³/ч" << endl;
}

// 3. Специальная функция для варианта 1: поиск насоса с макс. производительностью
Equipment find_max_flow_rate_pump(const vector<Equipment>& pumps) {
    // Предполагаем, что вектор не пустой (по условию задачи)
    Equipment max_pump = pumps[0];
    for (const auto& pump : pumps) {
        if (pump.flow_rate > max_pump.flow_rate) {
            max_pump = pump;
        }
    }
    return max_pump;
}

int main() {
    // 4. Создание массива (вектора) структур с данными
    vector<Equipment> equipment_list = {
        {1, "Насос центробежный ЦН-500", "Цех №1", 2018, 500.0},
        {2, "Насос шестеренчатый ШН-200", "Цех №2", 2020, 200.0},
        {3, "Насос вакуумный ВН-150", "Цех №3", 2019, 150.0},
        {4, "Насос дозировочный НД-80", "Цех №1", 2022, 80.0},
        {5, "Насос винтовой ВН-350", "Цех №2", 2021, 350.0}
    };

    // 5. Вывод исходного списка оборудования
    cout << "--- Исходный список насосного оборудования ---" << endl;
    for (const auto& item : equipment_list) {
        print_equipment(item);
    }
    cout << endl;

    // 6. Работа специальной функции
    Equipment best_pump = find_max_flow_rate_pump(equipment_list);

    // 7. Вывод результата
    cout << "--- Результат поиска ---" << endl;
    cout << "Насос с максимальной производительностью:" << endl;
    print_equipment(best_pump);
    cout << endl;

    return 0;
}
```

5. Результаты работы

Вывод программы:

```
--- Исходный список насосного оборудования ---
ID: 1 | Наименование: Насос центробежный ЦН-500 | Цех: Цех №1 | Год ввода: 2018 | Производительность: 500 м³/ч
ID: 2 | Наименование: Насос шестеренчатый ШН-200 | Цех: Цех №2 | Год ввода: 2020 | Производительность: 200 м³/ч
ID: 3 | Наименование: Насос вакуумный ВН-150 | Цех: Цех №3 | Год ввода: 2019 | Производительность: 150 м³/ч
ID: 4 | Наименование: Насос дозировочный НД-80 | Цех: Цех №1 | Год ввода: 2022 | Производительность: 80 м³/ч
```

ID: 5 | Наименование: Насос винтовой ВН-350 | Цех: Цех №2 | Год ввода: 2021 | Производительность: 350 м³/ч
--- Результат поиска ---
Насос с максимальной производительностью:
ID: 1 | Наименование: Насос центробежный ЦН-500 | Цех: Цех №1 | Год ввода: 2018 | Производительность: 500 м³/ч

Анализ результатов:

Алгоритм корректно перебрал все элементы массива и сравнил значения поля *flow_rate*. Максимальное значение (500 м³/ч) принадлежит объекту с ID 1, что и было выведено на экран.

6. Вывод

В ходе выполнения лабораторной работы:

- Была создана и использована структура *Equipment* для хранения данных о технологическом оборудовании, что позволило объединить разнотипную информацию (числовые идентификаторы, строки, год выпуска) в единую сущность.
- Освоены способы инициализации объектов структуры (при создании вектора).

Реализованы функции для работы со структурами:

- print_equipment (передача по константной ссылке для экономии памяти и защиты данных);
- find_max_flow_rate_pump (принимает вектор и возвращает структуру).

Успешно решена прикладная задача, характерная для нефтехимического производства - поиск оборудования с экстремальным значением параметра (максимальная производительность насоса).

Таким образом, цель работы достигнута, навык использования структур сформирован.

7. Ответы на контрольные вопросы

Контрольные вопросы (для допуска и защиты):

1. Объясните, что такое композиция данных и как структура реализует этот принцип.

Композиция - объединение разнотипных данных в одну сложную единицу. **Структура** реализует это через группировку полей в один тип Equipment*Equipment*, представляющий целостную сущность.

2. В какой области памяти (стек или куча) будет размещен массив ваших структур, созданный как локальная переменная в main()?

Сам объект вектора (equipment_list) - в **стеке** (локальная переменная). **Данные структур**, хранящиеся в векторе - в **куче** (динамическая память).

3. Предложите, как можно модифицировать программу, чтобы данные об оборудовании загружались из текстового файла. Как при этом должна измениться структура?

Добавить #include <fstream> и функцию liadFromFile(). Читать файл построчно, парсить значения (например, через запятую), заполнять вектор. **Структуру менять не нужно**, меняется только способ инициализации данных.

4. Почему в реальных проектах для подобных целей часто используют не структуры (struct), а классы (class)? В чем ключевое различие применительно к вашей задаче?

Инкапсуляция: в классе поля по умолчанию **приватные**, доступ через методы. **Контроль данных:** можно добавить проверки в сеттеры. **Гибкость:** легче расширять (наследование, полиморфизм) и поддерживать код. **Структуры** используют для простых "контейнеров" данных, **классы** - для сложной логики и защиты.