Федеральное государственное автономное образовательное учреждение высшего образования «Российский университет дружбы народов имени Патриса Лумумбы»

Факультет искусственного интеллекта

(наименование основного учебного подразделения (ОУП) – разработчика ОП ВО)

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C++ (ПРОДВИНУТЫЙ КУРС)

(наименование дисциплины/модуля)

Рекомендована МССН для направления подготовки/специальности:

02.04.02 ФУНДАМЕНТАЛЬНАЯ ИНФОРМАТИКА И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

(код и наименование направления подготовки/специальности)

Освоение дисциплины ведется в рамках реализации основной профессиональной образовательной программы высшего образования (ОП ВО):

УПРАВЛЕНИЕ ДАННЫМИ И ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

(наименование (профиль/специализация) ОП ВО)

2026 г.

1. ЦЕЛЬ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Дисциплина «Программирование на языке C++ (продвинутый курс)» входит в программу магистратуры «Управление данными и искусственный интеллект» по направлению 02.04.02 «Фундаментальная информатика и информационные технологии» и изучается в 1 семестре 1 курса. Дисциплину реализует Кафедра прикладного искусственного интеллекта. Дисциплина состоит из 3 разделов и 27 тем и направлена на изучение развитие профессиональных знаний и навыков эффективной разработки современных, надежных, многопоточных и высокопроизводительных приложений на C++, необходимых для решения сложных инженерных, научных, индустриальных и исследовательских задач в области искусственного интеллекта, анализа и обработки больших данных, компьютерного зрения, систем реального времени и высоконагруженных сервисов. Особое внимание уделяется современным аспектам языка (C++17/20), шаблонному программированию, интеграции с библиотеками AI/Big Data, производственной оптимизации, мультипоточности, распределенности и best-practices.
Целью освоения дисциплины является сформировать у студентов глубокое понимание синтаксиса и архитектурных возможностей C++ на современном уровне, научить реализовывать эффективные, безопасные и расширяемые программные решения для задач управления данными и высокопроизводительных вычислений, овладеть шаблонным проектированием, многопоточностью и интеграцией C++ с экосистемами искусственного интеллекта и центров обработки данных.

2. ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Освоение дисциплины «Программирование на языке C++ (продвинутый курс)» направлено на формирование у обучающихся следующих компетенций (части компетенций):

Таблица 2.1. Перечень компетенций, формируемых у обучающихся при освоении дисциплины (результаты освоения дисциплины)

Шифр	Компетенция	Индикаторы достижения компетенции (в рамках данной дисциплины)
УК-7	Способен: искать нужные источники информации и данные, воспринимать, анализировать, запоминать и передавать информацию с использованием цифровых средств, а также с помощью алгоритмов при работе с полученными из различных источников данными с целью эффективного использования полученной информации для решения задач; проводить оценку информации, ее достоверность, строить логические умозаключения на основании поступающих информации и данных	УК-7.3 Владеет навыками применения цифровых технологий и методов поиска, обработки, анализа, хранения и представления информации в области профессиональной деятельности;
ОПК-2	Способен применять компьютерные / суперкомпьютерные методы, современное программное обеспечение (в том числе отечественного производства) для решения задач профессиональной деятельности	ОПК-2.1 Знает основные положения и концепции в области программирования, архитектуру языков программирования, теории коммуникации, знает основную терминологию, знаком с перечнем ПО, включенного в Единый Реестр Российских программ; ОПК-2.2 Умеет анализировать типовые языки программирования, составлять программы; ОПК-2.3 Имеет практический опыт решения задач анализа, интеграции различных типов программного обеспечения, анализа типов коммуникации;

Шифр	Компетенция	Индикаторы достижения компетенции (в рамках данной дисциплины)
ПК-2	Способен проектировать, разрабатывать и поддерживать интегрированное программное обеспечение с использованием нейросетевых моделей и сквозных технологий искусственного интеллекта	ПК-2.2 Выбирает и моделирует архитектурные решений для реализации интегрированного программного обеспечения с использованием нейросетевых моделей и сквозных технологий искусственного интеллекта; ПК-2.3 Имеет навыки применения методов математического моделирования объектов и процессов, машинного обучения при разработке требований и проектировании программного обеспечения систем и моделей искусственного интеллекта;

3. МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ОП ВО

Дисциплина «Программирование на языке С++ (продвинутый курс)» относится к обязательной части блока 1 «Дисциплины (модули)» образовательной программы высшего образования. В рамках образовательной программы высшего образования обучающиеся также осваивают другие дисциплины и/или практики, способствующие достижению запланированных результатов освоения дисциплины «Программирование на языке С++ (продвинутый курс)».

Таблица 3.1. Перечень компонентов ОП ВО, способствующих достижению запланированных результатов освоения дисциплины

Шифр	Наименование компетенции	Предшествующие дисциплины/модули, практики*	Последующие дисциплины/модули, практики*
УК-7	Способен: искать нужные источники информации и данные, воспринимать, анализировать, запоминать и передавать информацию с использованием цифровых средств, а также с помощью алгоритмов при работе с полученными из различных источников данными с целью эффективного использования полученной информации для решения задач; проводить оценку информации, ее достоверность, строить логические умозаключения на основании поступающих информации и данных		Методы машинного обучения (продвинутый курс); Глубокое обучение для NLP; Генеративный искусственный интеллект; Преддипломная практика;
ОПК-2	Способен применять компьютерные / суперкомпьютерные методы, современное программное обеспечение (в том числе отечественного производства) для решения задач профессиональной деятельности		Технологическая (проектно-технологическая) практика (производственная); Методы машинного обучения (продвинутый курс); Глубокое обучение в компьютерном зрении; Методы оптимизации; Глубокое обучение для NLP;
ПК-2	Способен проектировать, разрабатывать и		Технологическая (проектно-технологическая)

Шифр	Наименование компетенции	Предшествующие дисциплины/модули, практики*	Последующие дисциплины/модули, практики*
	поддерживать интегрированное программное обеспечение с использованием нейросетевых моделей и сквозных технологий искусственного интеллекта		практика (производственная); Преддипломная практика; Методы машинного обучения (продвинутый курс); Глубокое обучение в компьютерном зрении; Машинное обучение на больших данных; Обучение с подкреплением; Генеративный искусственный интеллект; Искусственный интеллект в компьютерных играх**; Искусственный интеллект по отраслям**; Вайб-кодинг**;

* - заполняется в соответствии с матрицей компетенций и СУП ОП ВО

** - элективные дисциплины /практики

4. ОБЪЕМ ДИСЦИПЛИНЫ И ВИДЫ УЧЕБНОЙ РАБОТЫ

Общая трудоемкость дисциплины «Программирование на языке С++ (продвинутый курс)» составляет «4» зачетные единицы.

Таблица 4.1. Виды учебной работы по периодам освоения образовательной программы высшего образования для очной формы обучения.

Вид учебной работы	**ВСЕГО, ак.ч.**		**Семестр(-ы)**
			1
Контактная работа, ак.ч	*34*		*34*
Лекции (**ЛК**)	0		0
Лабораторные работы (**ЛР**)	0		0
Практические/семинарские занятия (**СЗ**)	34		34
Самостоятельная работа обучающихся, ак.ч.	*83*		*83*
Контроль (экзамен/зачет с оценкой), ак.ч.	*27*		*27*
Общая трудоемкость дисциплины ак.ч.	**ак.ч.**	**144**	**144**
	зач.ед.	**4**	**4**

5. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

*Таблица 5.1. Содержание дисциплины (модуля) по видам учебной работы**

Номер раздела	Наименование раздела дисциплины	Наименование темы		Содержание темы	Вид учебной работы*
Раздел 1	Современный C++: расширенные возможности языка и шаблонное программирование	1.1	Новые возможности C++17/20: auto, structured bindings, constexpr, smart pointers, move-семантика	Изучение современных возможностей стандартов C++17 и C++20, включая structured bindings для удобной распаковки данных, if constexpr для условной компиляции, новые типы std::optional, std::variant и std::any для безопасной работы с данными. Рассматриваются fold expressions для работы с вариативными шаблонами, концепты для ограничения шаблонных параметров, библиотека Ranges для функционального программирования и корутины для асинхронного кода.	
		1.2	Шаблоны функций и классов: SFINAE, enable_if, метапрограммирование	Основы создания шаблонных функций, механизмы специализации шаблонов для конкретных типов, автоматический вывод типов компилятором. Изучение SFINAE (Substitution Failure Is Not An Error) для управления перегрузкой функций, вариативные шаблоны для функций с произвольным числом параметров, техника perfect forwarding для эффективной передачи аргументов без копирования.	
		1.3	Современная стандартная библиотека: STL, многомерные контейнеры, алгоритмы	Создание обобщённых классов, работающих с различными типами данных. Частичная и полная специализация шаблонов классов, использование параметров по умолчанию, особенности наследования в шаблонных классах. Применение friend функций и классов в контексте шаблонов для предоставления доступа к приватным членам.	
		1.4	Практика реализации move-семантики и автоматического управления ресурсами	Глубокое изучение Standard Template Library: контейнеры (vector, list, map, set, unordered_map и другие), различные категории итераторов и их использование. Алгоритмы STL для обработки данных (sort, find, transform, accumulate), функциональные объекты и лямбда-выражения для настройки поведения алгоритмов, адаптеры контейнеров для изменения интерфейса.	
		1.5	Разработка обобщённых контейнеров и алгоритмов с использованием шаблонов	Изучение проверенных практик разработки на C++: идиома RAII для автоматического управления ресурсами, правила Zero/Three/Five для корректной работы с копированием и перемещением объектов. Copy-and-swap идиома для безопасного присваивания, Pimpl для скрытия деталей реализации, CRTP для статического полиморфизма, техника type erasure для динамического полиморфизма без виртуальных функций.	
		1.6	Задачи на применение новых возможностей STL (std::optional, std::variant, std::any) в прикладных программах	Практическая разработка контейнера с нуля: проектирование интерфейса, совместимого с STL, эффективное управление памятью с использованием аллокаторов. Реализация собственных итераторов, обеспечение исключительной безопасности (basic, strong, nothrow guarantee), оптимизация производительности и обеспечение полной совместимости с алгоритмами STL.	
		1.7	Разбор типовых ошибок при использовании новых возможностей языка	Проектирование переиспользуемых обобщённых компонентов, техники метапрограммирования для вычислений на этапе компиляции. Использование type traits для получения информации о типах, compile-time вычисления для оптимизации, применение constexpr функций и переменных для выполнения кода во время компиляции.	СЗ

		1.8	Кейс-стади: шаблонный дизайн для библиотек данных и AI-платформ	Анализ реальных примеров шаблонного дизайна из стандартной библиотеки STL, детальный разбор архитектуры библиотек Boost. Изучение классических шаблонов проектирования, адаптированных для использования с template, решение практических задач с применением шаблонного метапрограммирования.	СЗ

Номер раздела	Наименование раздела дисциплины	Наименование темы		Содержание темы	Вид учебной работы*
		1.9	Групповая работа: обсуждение границ применимости метапрограммирования и generics	Совместная разработка проекта в команде с применением изученных техник. Практики code review для повышения качества кода, эффективное использование систем контроля версий (Git), создание качественной документации кода с использованием Doxygen или аналогичных инструментов.	СЗ
Раздел 2	Многопоточность, параллелизм и высокопроизводительные вычисления	2.1	Средства многопоточности: std::thread, std::async, std::future, std::mutex, условные переменные	Создание и управление потоками выполнения с использованием std::thread, передача параметров в поток, join и detach операции. Асинхронное выполнение задач через std::async с различными политиками запуска (launch::async, launch::deferred). Механизмы получения результатов из потоков с помощью std::future и std::shared_future, передача данных и исключений между потоками через std::promise. Примитивы синхронизации: различные типы мьютексов (std::mutex, std::recursive_mutex, std::shared_mutex, std::timed_mutex) для защиты разделяемых данных. RAII-обёртки для автоматического управления блокировками: std::lock_guard, std::unique_lock, std::shared_lock. Условные переменные (std::condition_variable) для координации работы потоков, паттерны producer-consumer, wait-notify механизмы.	
		2.2	Современные подходы к параллельной обработке данных: thread pool, lock-free структуры	Реализация пула потоков для эффективного переиспользования ресурсов и уменьшения накладных расходов на создание потоков. Проектирование thread-safe очередей задач, балансировка нагрузки между потоками, graceful shutdown. Разработка lock-free алгоритмов и структур данных с использованием атомарных операций std::atomic. Compare-and-swap (CAS) операции для реализации lock-free списков, стеков, очередей. ABA проблема и способы её решения. Создание wait-free структур данных для максимальной производительности и гарантированного прогресса каждого потока. Memory ordering models: relaxed, acquire, release, acq_rel, seq_cst и их влияние на производительность и корректность.	
		2.3	Оптимизация вычислений: cache-friendly код, SIMD, параллельные алгоритмы STL	Понимание архитектуры кэша процессора (L1, L2, L3) и его критического влияния на производительность. Концепция cache line (обычно 64 байта), проблема false sharing при многопоточном доступе к близко расположенным данным и способы её избежать через выравнивание и padding. Выравнивание данных (alignas, alignof) для оптимального размещения в памяти. Паттерны доступа к памяти: преимущества последовательного доступа над случайным, оптимизация расположения данных (data-oriented design, structure of arrays vs array of structures). Векторизация вычислений с использованием SIMD инструкций для одновременной обработки нескольких элементов данных. Intrinsic функции для прямого доступа к SIMD инструкциям процессора (SSE, AVX, AVX-512), автоматическая векторизация компилятором и способы её улучшения через hints. Параллельные алгоритмы STL (C++17) с execution policies: std::execution::seq (последовательное выполнение), std::execution::par (параллельное), std::execution::par_unseq (параллельное векторизованное). Применение к стандартным алгоритмам: std::sort, std::transform, std::reduce.	

		2.4	Реализация многопоточной задачи: вычисление агрегации или поиск по сегментам	Практическая реализация параллельного вычисления агрегатных функций (сумма, среднее, максимум, минимум) на больших массивах данных. Разделение данных на сегменты для обработки независимыми потоками, выбор оптимального размера сегмента с учётом числа ядер процессора. Параллельный поиск элементов в отсортированных и неотсортированных данных, реализация map-reduce паттерна. Сбор	

Номер раздела	Наименование раздела дисциплины	Наименование темы		Содержание темы	Вид учебной работы*
				и комбинирование результатов от потоков, минимизация синхронизации. Измерение ускорения (speedup) и эффективности (efficiency) параллельной реализации, закон Амдала и его практические следствия.	
		2.5	Разработка parallel pipline: обработка большого массива с синхронизацией	Создание конвейерной (pipeline) архитектуры для многоэтапной обработки данных, где каждый этап выполняется отдельным потоком или группой потоков. Проектирование stage-based обработки: чтение данных → обработка → запись результатов. Синхронизация между этапами конвейера с использованием очередей, семафоров, условных переменных. Балансировка производительности этапов для избежания bottleneck, buffering между этапами. Обработка больших массивов данных потоковым образом (streaming) для минимизации использования памяти. Graceful degradation при ошибках на одном из этапов, exception propagation через pipeline.	
		2.6	Оптимизация вычисления с использованием возможностей многопоточности и SIMD	Комбинированное применение многопоточности и SIMD для максимальной производительности. Разделение работы между потоками таким образом, чтобы каждый поток мог эффективно использовать векторизацию. Оптимизация циклов для автоматической векторизации компилятором: устранение зависимостей данных, выравнивание массивов, hints компилятору. Manual vectorization с использованием intrinsics для критичных участков кода. Профилирование с использованием VTune, perf, gprof для выявления узких мест. Анализ assembly кода для проверки векторизации. Trade-offs между читаемостью кода и производительностью.	
		2.7	Анализ типичных ошибок синхронизации и гонок данных на примерах	Детальное изучение распространённых проблем многопоточного программирования на конкретных примерах кода. Race conditions (состояние гонки) при одновременном чтении и записи разделяемых данных без должной синхронизации, использование ThreadSanitizer для обнаружения. Deadlock (взаимная блокировка) при циклических зависимостях между блокировками, std::lock для атомарного захвата нескольких мьютексов, иерархия блокировок. Livelock, когда потоки активны но не прогрессируют, постоянно реагируя друг на друга. Starvation (голодание) при несправедливом распределении ресурсов между потоками. Priority inversion в системах реального времени и протоколы её решения (priority inheritance, priority ceiling). Использование инструментов для обнаружения: Helgrind, DRD, ThreadSanitizer.	СЗ
		2.8	Анализ типичных ошибок синхронизации и гонок данных на примерах	Продолжение углубленного анализа проблем многопоточности. Детальный разбор реальных багов из production кода, их причин и способов предотвращения. Использование formalьных методов для верификации многопоточного кода. Паттерны безопасного многопоточного программирования: immutability, thread confinement, synchronization primitives. Code review checklist для многопоточного кода. Практика в создании unit тестов для воспроизведения race conditions. Stress testing многопоточных приложений.	СЗ

		2.9	Практикум по анализу кода: оценка скорости, устранение bottleneck, обсуждение trade-off производительности и сложности	Практическая работа с реальным кодом: профилирование приложения для выявления горячих точек (hot spots). Использование профайлеров (perf, VTune, Instruments, Visual Studio Profiler) для анализа CPU time, cache misses, branch mispredictions. Flame graphs для визуализации времени выполнения. Определение bottleneck: CPU-bound vs IO-bound vs memory-bound задачи. Применение оптимизаций: алгоритмические улучшения, использование более эффективных структур данных, кэширование результатов, параллелизация. Измерение эффекта оптимизаций через бенчмарки.	СЗ

Номер раздела	Наименование раздела дисциплины	Наименование темы		Содержание темы	Вид учебной работы*
				Обсуждение trade-offs: производительность vs читаемость кода, производительность vs поддерживаемость, оптимизация по времени vs оптимизация по памяти. Когда оптимизировать и когда остановиться (premature optimization). Документирование принятых решений и их обоснование.	
Раздел 3	Интеграция C++ в сложные программные системы AI, Big Data, HPC	3.1	C++ и взаимодействие с внешними библиотеками: OpenCV, PyBind, TensorRT, protobuf	Подключение и настройка внешних библиотек в C++ проекте через CMake, pkg-config, vcpkg, Conan. Работа с OpenCV для обработки изображений и компьютерного зрения: загрузка/сохранение изображений, фильтрация, трансформации, детектирование объектов, интеграция с DNN модулями OpenCV. Создание Python биндингов для C++ кода с помощью PyBind11: экспорт функций, классов, STL контейнеров в Python, управление памятью, обработка исключений. Ускорение ML inference с NVIDIA TensorRT: оптимизация моделей, квантизация, создание engine, интеграция в C++ приложение. Protocol Buffers для сериализации структурированных данных: определение .proto файлов, генерация C++ кода, эффективная сериализация/десериализация, версионирование протоколов.	
		3.2	Использование C++ для взаимодействия с БД, распределёнными и обработчиками потоков данных (gRPC, Boost.Asio, sockets, SQL/NoSQL bindings)	Работа с реляционными SQL базами данных через native драйверы (libpq для PostgreSQL, MySQL Connector/C++), ODBC для универсального доступа, ORM библиотеки (ODB, SOCI). Реализация connection pooling для эффективного переиспользования соединений к БД. Асинхронная работа с БД для неблокирующего ввода-вывода, prepared statements для защиты от SQL injection. Интеграция с NoSQL базами: MongoDB (mongocxx driver), Redis (hiredis, redis-plus-plus), Cassandra (DataStax C++ driver). Разработка распределённых систем с использованием gRPC: определение сервисов в .proto, генерация серверного и клиентского кода, streaming RPC, authentication и encryption. Boost.Asio для асинхронного сетевого программирования: io_context, async operations, coroutines. Низкоуровневая работа с сокетами: TCP/UDP, non-blocking IO, epoll/kqueue для масштабируемых серверов.	
		3.3	Организация CI/CD, тестирования, покрытие кода и практическое развертывание сервисов на C++	Настройка Continuous Integration с Jenkins, GitLab CI, GitHub Actions, Travis CI для автоматической сборки C++ проектов при каждом коммите. Конфигурация build matrix для тестирования на различных платформах (Linux, Windows, macOS) и компиляторах (GCC, Clang, MSVC). Автоматическое тестирование: unit тесты с Google Test, Catch2, Boost.Test; integration тесты; end-to-end тесты. Измерение покрытия кода (code coverage) с gcov, lcov, coveralls.io. Статический анализ кода: clang-tidy для выявления багов и code smells, cppcheck, PVS-Studio. Dynamic analysis: Valgrind для поиска утечек памяти, AddressSanitizer, ThreadSanitizer. Continuous Deployment: автоматическое развёртывание на staging/production, Docker контейнеризация C++ приложений, Kubernetes для оркестрации. Мониторинг производительности в production: логирование, метрики, трейсинг.	

		3.4	Интеграция C++-модуля с Python (PyBind, SWIG): встраивание в пайплайн AI/ML	Детальное изучение создания расширений Python на C++ для ускорения критичных участков кода в ML пайплайнах. PyBind11: биндинг функций, классов, операторов, STL контейнеров, NumPy arrays (zero-copy взаимодействие), обработка GIL. SWIG как альтернатива для генерации биндингов для множества языков. Эффективная передача больших массивов данных между Python и C++ без копирования через buffer protocol. Корректная обработка исключений на границе языков, mapping C++ exceptions в Python. Управление памятью при взаимодействии Python garbage collector и C++ RAII,	

Номер раздела	Наименование раздела дисциплины	Наименование темы		Содержание темы	Вид учебной работы*
				shared_ptr интеграция. Встраивание C++ модулей в типичный ML пайплайн: предобработка данных на C++, вызов inference, постобработка. Профилирование Python+C++ приложений для определения оптимального разделения логики. Упаковка и распространение Python пакетов с C++ расширениями через setuptools, wheels.	
		3.5	Разработка сетевого или сервисного модуля с асинхронным взаимодействием	Проектирование и реализация высокопроизводительного асинхронного сервера на C++. Выбор модели асинхронности: callbacks, futures/promises, coroutines (C++20). Реализация с Boost.Asio или standalone Asio: io_context для event loop, async_read/async_write для неблокирующего IO, strand для синхронизации. Создание HTTP/HTTPS сервера: парсинг запросов, роутинг, формирование ответов, поддержка WebSocket. REST API разработка на C++. Масштабируемость через thread pool для обработки запросов, reactor/proactor паттерны. Connection management: keep-alive, timeout, graceful shutdown. SSL/TLS интеграция через OpenSSL. Load balancing и reverse proxy considerations. Разработка собственного протокола поверх TCP для специфических задач с высокими требованиями к производительности.	
		3.6	Настройка юнит- и интеграционных тестов, сборки и деплоя C++-проекта для исследовательской или индустриальной среды	Организация тестирования C++ проекта на всех уровнях. Unit тесты: выбор фреймворка (Google Test, Catch2, Doctest), написание тестов для функций и классов, моки и стабы с Google Mock, test fixtures, параметризованные тесты. Integration тесты: тестирование взаимодействия компонентов, тестовые БД и окружение, Docker для изолированного тестового окружения. End-to-end тесты для проверки всей системы. Test-driven development (TDD) практики. Организация системы сборки с CMake: out-of-source builds, конфигурационные опции, поиск зависимостей, install правила, CPack для создания пакетов. Continuous Integration настройка для автоматического запуска тестов. Deployment стратегии: binary distribution, containerization с Docker, package management (apt, yum, brew). Настройка для исследовательской среды: воспроизводимость сборок, документация для коллег. Индустриальная среда: compliance требования, security scanning, release management.	
		3.7	Разбор успешных open-source кейсов: архитектуры C++-сервисов для AI и Big Data	Анализ архитектуры реальных успешных проектов с открытым исходным кодом. TensorFlow: C++ core для вычислений, Python frontend, graph execution model, distributed training. PyTorch/LibTorch: eager execution, autograd engine на C++, JIT compilation. Apache Arrow: колоночный формат данных в памяти, zero-copy взаимодействие между языками, compute kernels на C++. ClickHouse: OLAP СУБД на C++, векторизация запросов, распределённая архитектура. Redis: однопоточная архитектура с event loop, модульная система. Анализ решений: почему выбран C++, какие паттерны используются, как организовано управление памятью, многопоточность, обработка ошибок. Lessons learned из этих проектов, применимые к собственной разработке.	СЗ

		3.8	Обсуждение стратегий интеграции и миграции между языками (C++, Python, C#, Java)	Стратегии организации polyglot проектов, использующих несколько языков программирования. C++ как performance-critical core с высокоуровневыми обёртками на Python/C#/Java для удобства использования. Foreign Function Interface (FFI): C ABI как универсальный интерфейс, создание C-совместимых экспортов из C++. JNI для интеграции C++ и Java, C++/CLI для взаимодействия с .NET. Микросервисная архитектура для изоляции компонентов на разных языках, взаимодействие через REST API, gRPC, message queues. Миграция legacy кода с C++ на современные языки: когда	СЗ

Номер раздела	Наименование раздела дисциплины	Наименование темы		Содержание темы	Вид учебной работы*
				имеет смысл, постепенная миграция модуль за модулем, сохранение критичной по производительности логики на C++. Обратная миграция: переписывание bottleneck с Python на C++. Inter-process communication для взаимодействия процессов на разных языках: shared memory, pipes, sockets. Управление зависимостями в polyglot проектах.	
		3.9	Совместный анализ CI/CD пайплайнов, тестирования и развертывания в современных инфраструктурах	Групповой разбор реальных CI/CD конфигураций для C++ проектов. Дизайн эффективного пайплайна: стадии build, test, analyze, package, deploy. Оптимизация времени сборки: кэширование зависимостей, ccache для кэширования компиляции, distributed builds, precompiled headers. Параллельная сборка и тестирование. Matrix builds для множества конфигураций. Artifacts management: хранение бинарей, debug symbols. Deployment в различные окружения: bare metal, virtual machines, containers (Docker, Podman), Kubernetes. Infrastructure as Code (Terraform, Ansible) для воспроизводимого деплоя. Blue-green deployments, canary releases для безопасного обновления production. Rollback стратегии при проблемах. Мониторинг CI/CD метрик: build success rate, build time trends, test flakiness. Security в CI/CD: secret management, vulnerability scanning, signed artifacts. Best practices и anti-patterns в CI/CD для C++ проектов.	СЗ

* - заполняется только по ОЧНОЙ форме обучения: ЛК – лекции; ЛР – лабораторные работы; СЗ – практические/семинарские занятия.

6. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Таблица 6.1. Материально-техническое обеспечение дисциплины

Тип аудитории	Оснащение аудитории	Специализированное учебное/лабораторное оборудование, ПО и материалы для освоения дисциплины (при необходимости)
Лекционная	Аудитория для проведения занятий лекционного типа, оснащенная комплектом специализированной мебели; доской (экраном) и техническими средствами мультимедиа презентаций.	
Компьютерный класс	Компьютерный класс для проведения занятий, групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации, оснащенная персональными компьютерами (в количестве [Параметр] шт.), доской (экраном) и техническими средствами мультимедиа презентаций.	
Семинарская	Аудитория для проведения занятий семинарского типа, групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации, оснащенная комплектом специализированной мебели и техническими средствами мультимедиа презентаций.	
Для самостоятельной работы	Аудитория для самостоятельной работы обучающихся (может использоваться для проведения семинарских занятий и консультаций), оснащенная комплектом специализированной мебели и компьютерами с доступом в ЭИОС.	

* - аудитория для самостоятельной работы обучающихся указывается **ОБЯЗАТЕЛЬНО**!

7. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Основная литература:

1. Немцова Тамара Игоревна, Голова Светлана Юрьевна, Терентьев Алексей Игоревич. Программирование на языке высокого уровня. Программирование на языке С++. учебное пособие / под редакцией Л.Г. Гагариной [Электронный ресурс]. - М.: ИНФРА-М, 2023. 512 с. ISBN 978-5-16-013214-3 URL: https://mega.rudn.ru/MegaPro/UserEntry?Action=Link_FindDoc&id=508904&idb=0

2. Панкратов Александр Серафимович, Салпагаров Солтан Исмаилович. Технология программирования на языке С++: динамические структуры, объекты, классы. учебное пособие [Электронный ресурс]. – М.: РУДН, 2021. 73 с. ISBN 978-5-209-10906-8 URL: https://mega.rudn.ru/MegaPro/UserEntry?Action=Link_FindDoc&id=504558&idb=0

Дополнительная литература:

1. Огнева, М. В. Программирование на языке С++: практический курс: учебник для вузов / М. В. Огнева, Е. В. Кудрина, А. А. Казачкова. — 2-е изд., перераб. и доп. — Москва: Издательство Юрайт, 2025. — 342 с. — (Высшее образование). — ISBN 978-5-534-18949-0. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/563618 (дата обращения: 14.05.2025)

2. Немцова, Т. И. Программирование на языке высокого уровня. Программирование на языке С++: учебное пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев ; под ред. Л.Г. Гагариной. — Москва: ФОРУМ: ИНФРА-М, 2024. — 512 с. + Доп. материалы [Электронный ресурс]. — (Среднее профессиональное образование). - ISBN 978-5-8199-0699-6. - Текст: электронный. - URL: https://znanium.com/catalog/product/2083383

Ресурсы информационно-телекоммуникационной сети «Интернет»:

1. ЭБС РУДН и сторонние ЭБС, к которым студенты университета имеют доступ на основании заключенных договоров
   - Электронно-библиотечная система РУДН – ЭБС РУДН https://mega.rudn.ru/MegaPro/Web
   - ЭБС «Университетская библиотека онлайн» http://www.biblioclub.ru
   - ЭБС «Юрайт» http://www.biblio-online.ru
   - ЭБС «Консультант студента» www.studentlibrary.ru
   - ЭБС «Знаниум» https://znanium.ru/
2. Базы данных и поисковые системы
   - Sage https://journals.sagepub.com/
   - Springer Nature Link https://link.springer.com/
   - Wiley Journal Database https://onlinelibrary.wiley.com/
   - Наукометрическая база данных Lens.org https://www.lens.org

Учебно-методические материалы для самостоятельной работы обучающихся при освоении дисциплины/модуля:*

1. Курс лекций по дисциплине «Программирование на языке C++ (продвинутый курс)».

* - все учебно-методические материалы для самостоятельной работы обучающихся размещаются в соответствии с действующим порядком на странице дисциплины **в ТУИС**!

РАЗРАБОТЧИКИ

	Подолько П.М.
Должность	Фамилия И.О

РУКОВОДИТЕЛЬ БУП

Заведующий кафедрой	Подолько П.М.
Должность	Фамилия И.О

РУКОВОДИТЕЛЬ ОП ВО

Заведующий кафедрой прикладного искусственного интеллекта	Подолько П.М.
Должность	Фамилия И.О