Основы кибернетики и вычислительной техники

Краткое содержание: Данная лекция посвящена организационным вопросам курса и введению в базовые понятия кибернетики и вычислительной техники. Обсуждается порядок выполнения и сдачи лабораторных работ через систему LMS, критерии оценивания и важность самостоятельного выполнения заданий. Введено определение кибернетики как науки об управлении и обратной связи, проиллюстрирован принцип обратной связи на примерах управления лошадью и беспилотным летательным аппаратом. Рассмотрено место искусственного интеллекта в кибернетике: искусственный интеллект определён как алгоритмы, заменяющие некоторые когнитивные функции человека, при этом подчеркнуто, что по сути он представляет собой набор вычислительных алгоритмов (например, перемножение матриц). Лекция акцентирует внимание на необходимости фундаментальных знаний для программиста – алгоритмов, структур данных, устройства вычислительных систем, а также базовой математической подготовки (матанализ, линейная алгебра, теория вероятностей, дискретная математика). Отдельно затрагивается тема использования современных нейросетевых моделей: студентам рекомендуется применять их для поиска и структурирования информации, однако при освоении навыков программирования важно тренироваться самостоятельно, чтобы не утратить способность к алгоритмическому мышлению. Наконец, вводятся основы формальной логики и булевой алгебры – показывается, что вся сложная вычислительная техника и программное обеспечение основаны на пяти базовых элементах (истина, ложь, логические операции И, ИЛИ, НЕ), и объясняется понятие формальной системы (алфавит, формулы, аксиомы, правила вывода) как основы строгого математического описания логики.

Организация курса и лабораторные работы

Курс «Основы кибернетики и вычислительной техники» предполагает выполнение серии лабораторных работ, которые студенты должны сдавать через систему управления обучением (LMS). Всего запланировано четыре лабораторных задания, причем для каждого предусмотрено около десяти вариантов – это сделано, чтобы у разных студентов были разные условия и тем самым снизить вероятность простого копирования решений. Каждое выполненное задание требуется загрузить в LMS отдельным файлом в соответствующий раздел курса. Если работа не прикреплена в LMS должным образом, преподаватель не сможет её проверить и оценить.

Для успешного прохождения курса необходимо выполнить все лабораторные работы в установленные сроки. Дедлайн для сдачи всех заданий ориентировочно приходится на конец декабря (ещё до начала экзаменационной сессии в феврале). Это связано с большим количеством студентов (порядка 200 человек в потоке и нескольких потоков у преподавателя) – работ должно хватить времени проверить. При сдаче важно строго соблюдать требования к формату файлов. Например, лабораторные работы в данном курсе выполняются в программе Logisim (цифровой логический симулятор) с сохранением схем в формате .circ. Система LMS не принимает файлы с таким расширением, поэтому установлено правило: результаты работы в Logisim (схемы и т.д.) необходимо вставить в отчёт в формате Microsoft Word (.docx) – например, сделать скриншоты схем и поместить их в документ. Затем уже этот файл .docx следует прикрепить в LMS. Если попытаться прикрепить непосредственно .circ, система выдаст ошибку. Студентам настоятельно рекомендуется не откладывать загрузку работы на последний момент, чтобы успеть проверить совместимость формата и исправить возможные технические проблемы заранее. Если файл не прикрепился, а дедлайн уже прошёл (например, в полночь дня сдачи), работа считаться не будет.

Каждая лабораторная работа оценивается после проверки преподавателем. Важно выполнить работы самостоятельно: хотя обмен идеями не запрещён, прямое копирование решений у однокурсников строго пресекается. Преподаватель отмечает, что при проверке легко распознать типичные «шаблонные» решения и случаи списывания – в таких ситуациях работы не засчитываются. Чтобы избежать соблазна списывания, рекомендуется «списывать правильно у преподавателя», то есть опираться на алгоритмы и рекомендации, данные на лекциях, вместо того чтобы копировать чужой готовый код. В результате успешного выполнения лабораторных заданий формируется и экзаменационная оценка по предмету: если студент корректно выполнил хотя бы две лабораторные работы, ему гарантируется удовлетворительная оценка («3»); если сделаны все четыре работы, можно рассчитывать как минимум на оценку «4» (хорошо). Для получения отличной оценки («5») требуется продемонстрировать полное освоение материала – фактически правильное выполнение всего массива заданий курса в установленные сроки (преподаватель упоминал, что при условии выполнения пяти условных частей заданий ставится «5»). Другими словами, чтобы претендовать на максимальную оценку, необходимо выполнить все лабораторные работы верно и полностью.

Взаимодействие на практических занятиях и вопросы

Формат курса предполагает, что основные алгоритмы выполнения заданий преподаватель объясняет на лекциях, а традиционных «практических занятий» в аудитории не проводится. Однако от студентов ожидается самостоятельная реализация обсуждённых на лекции решений. Если при выполнении задания возникают вопросы, на которые не было ответа в лекционном материале, их можно задать преподавателю в рамках отведённого времени. Для этого предусмотрены сессии обратной связи: во время условных практических занятий или консультаций студенты могут обратиться к преподавателю (например, через поднятие руки в видеоконференции или по корпоративной почте). Важно предварительно убедиться, что вопрос действительно существенный: сначала рекомендуется пересмотреть запись лекции и изучить предоставленные материалы самостоятельно. Преподаватель подчёркивает, что «глупые» вопросы (возникающие от невнимательности или нежелания приложить усилия) рассматриваться не будут, тогда как умные вопросы, свидетельствующие о попытке вникнуть в суть и разобраться, приветствуются. Иными словами, если что-то действительно непонятно по существу материала (например, недостаточно ясно объяснён принцип или шаг алгоритма), преподаватель готов это дополнительно пояснить на консультации. Но если студент пропустил объяснение или не удосужился прочитать методические указания, и из-за этого путается в задании, от него ожидается сперва самостоятельное изучение – такие пробелы не являются основанием для индивидуального разбора. Подобный подход дисциплинирует и приучает к самостоятельности, необходимой в профессиональной деятельности.

Еще один технический момент: в LMS обычно можно заменить прикреплённый файл до истечения срока сдачи. Студент имеет возможность загрузить файл, проверить, что он успешно прикрепился, и даже удалить/заменить его, пока не нажата кнопка окончательной сдачи (или пока дедлайн не наступил). Однако после окончательной отправки или закрытия дедлайна работа считается поданной, и отозвать или исправить её уже нельзя. Поэтому рекомендуется загрузить работу заранее и убедиться, что всё в порядке, вместо того чтобы выкладывать в последний момент без проверки.

Основы кибернетики: управление и обратная связь

Перед тем как перейти к техническим аспектам вычислительной техники, курс даёт обзор понятия кибернетики. Кибернетика – это наука об управлении и обратных связях. Такое определение было дано основателем этой науки Норбертом Винером, автором книги «Кибернетика, или управление и связь в животном и машине». Иными словами, кибернетика изучает, как в системах – будь то биологические организмы, технические устройства или социальные структуры – осуществляется целенаправленное управление посредством получения и переработки информации о состоянии системы (обратной связи).

Для иллюстрации принципа управления с обратной связью можно рассмотреть простой образ: кучер и лошадь. Кучер управляет лошадью, направляя её движение – например, отдавая команды движением вожжей или голосом. Лошадь может ускоряться, поворачивать, останавливаться по командам кучера. Однако чтобы управление было эффективным и цель достигнута (например, чтобы лошадь везла повозку по заданному маршруту и не сбилась с дороги), одного лишь «приказа» недостаточно – необходима обратная связь. Кучер должен получать информацию о фактическом поведении лошади и окружающей обстановке: он наблюдает, туда ли движется лошадь, не возникает ли препятствий на пути. Используя зрение и другие чувства, кучер постоянно сравнивает текущую ситуацию с желаемой (запланированным маршрутом) и на основе этого корректирует действия: например, натягивает вожжи, если лошадь свернула не туда, замедляет или ускоряет бег, поворачивает вправо или влево. Зрение кучера в этой системе – это и есть канал обратной связи, позволяющий сопоставлять команду и результат и при необходимости вносить корректировки.

Более современным примером является система автопилота беспилотного летательного аппарата (дрона). Здесь цель управления – удерживать аппарат на заданной высоте. Управляющее воздействие осуществляется с помощью рулевого элемента – например, руля высоты (закрылка). Если изменить угол этого руля, по законам аэродинамики изменится подъемная сила, и дрон поднимется или опустится. Но чтобы точно держать нужную высоту полёта, автопилоту нужна информация о текущей высоте – иначе любая разовая команда (например, отклонить руль высоты на определённый угол) может оказаться недостаточной или избыточной из-за влияния множества факторов (порывы ветра, изменение массы дрона при расходе топлива и т.д.). Поэтому в аппарате установлен датчик высоты (например, барометрический высотомер), который постоянно измеряет текущую высоту, сравнивает её с заданной и вычисляет отклонение. Если дрон оказался ниже требуемого уровня, система подаст сигнал увеличить угол закрылка (поднять нос) до тех пор, пока разница не будет сведена к нулю; если же аппарат поднялся выше цели, управляющая система скорректирует руль высоты в обратную сторону, чтобы снизить высоту. Такой цикл – «сравнить с целью – отклониться/подкорректировать – снова измерить» – повторяется непрерывно, образуя замкнутый контур управления с обратной связью. В контуре присутствуют: объект управления (беспилотник), цель управления (заданная высота полёта), сенсор (датчик, сообщающий текущее значение высоты), ошибка (разность между текущей и требуемой высотой) и управляющий сигнал (команда рулю высоты). На основе ошибки вырабатывается управляющее воздействие, стремясь минимизировать эту ошибку. В памяти бортового компьютера заложен алгоритм (формулы, дифференциальные уравнения), определяющий, насколько именно нужно отклонить руль при том или ином отклонении высоты, с учётом инерционности аппарата, внешних условий и пр. Таким образом, система автопилота автоматически поддерживает высоту, используя принцип обратной связи по аналогии с действиями кучера, лишь формализованными и запрограммированными.

Принцип управления с обратной связью универсален и применяется во множестве сфер. От бытовых приборов (например, робот-пылесос, автоматически передвигающийся по комнате и корректирующий свой маршрут при обнаружении препятствий) до промышленных комплексов (автоматизированные заводские линии, регулирующие технологические параметры) – всюду мы видим одну и ту же схему: есть объект управления, есть цель (заданный режим работы), есть датчики, оценивающие текущее состояние, есть вычислитель, сравнивающий состояние с целью и вычисляющий управляющие воздействия, и есть исполнительные механизмы, которые эти воздействия реализуют. Кибернетика занимается изучением и построением таких систем.

Искусственный интеллект в контексте кибернетики

В современном мире кибернетика охватывает не только классические задачи автоматического регулирования, но и замену ряда когнитивных функций человека алгоритмами. Первым шагом в этом направлении можно считать появление калькуляторов: ручной счёт и математические вычисления, которые раньше выполнял человек в уме или на бумаге, были переданы устройству, способному быстро складывать, умножать и т.д. Сейчас под искусственным интеллектом (ИИ) понимают совокупность алгоритмов, способных решать задачи, традиционно требовавшие человеческого интеллекта: распознавание образов (зрение), понимание и генерация речи и текста, принятие решений, прогнозирование событий и т.п. В лекции предложено простое определение: искусственный интеллект – это алгоритмы, которые частично заменяют когнитивные способности человека. Например, уже существуют программы, умеющие воспринимать зрительные образы (анализировать изображения и видео), «слышать» и понимать речь, генерировать осмысленные тексты, прогнозировать сложные процессы, подбирать оптимальные решения из множества вариантов. Хотя возможности таких систем впечатляют, важно понимать, что под капотом они представляют собой математические модели и вычислительные процессы. По сути, любой современный ИИ работает на уровне элементарных операций, выполненных очень быстро: например, многие модели машинного обучения сводятся к операциям перемножения матриц и применению несложных функций, реализованных аппаратно на уровне процессоров и графических ускорителей. Таким образом, термин «интеллект» в ИИ условен – фактически это набор сложных, но строго определённых алгоритмов и статистических моделей. Тем не менее, изучение и разработка таких алгоритмов – важнейшая часть современной кибернетики.

Содержание курса и учебные материалы

За ограниченное время курса невозможно детально охватить все области кибернетики и информатики, поэтому программа сфокусирована на базовых вещах, которые должен знать любой специалист в области программирования. Преподаватель сравнивает эти знания с азбукой или таблицей умножения: не зная их, сложно дальше постигать предмет. Планируется дать именно фундаментальные основы – своего рода «латинский алфавит» компьютерных наук, минимальный набор понятий и принципов, без которых трудно понимать устройство программно-аппаратных систем.

Чтобы студенты могли при желании углубиться в предмет, преподаватель подготовил подборку литературы. В частности, он рекомендовал несколько книг. Во-первых, Чарльз Петцольд, «Код. Тайный язык информатики» – популярная научно-познавательная книга, доступно и увлекательно рассказывающая о том, как устроены компьютеры на самом базовом уровне. Автор начинает с простых примеров (вплоть до электрического фонарика и азбуки Морзе) и шаг за шагом подводит читателя к пониманию принципов работы процессора. В книге нет сложных формул и она подходит даже тем, кто считает себя «чайником» в информатике: изложение ведётся простым языком, с историческими примерами и юмором. Преподаватель настоятельно советует эту книгу тем, у кого мало исходных знаний, – она позволит получить удовольствие от обучения и устранить пробелы.

Во-вторых, Харрис Д., Харрис С., «Цифровая схемотехника и архитектура компьютера» – фундаментальный учебник, используемый, например, в MIT для изучения архитектуры компьютеров. В нём изложены основы схемотехники, принципов работы цифровых устройств и компьютерных архитектур. Именно на основе этой книги во многом построены сами лекции курса. Однако объём издания около 760 страниц, поэтому на лекциях рассматриваются лишь первые главы (порядка 60 страниц), самые базовые идеи. Тем не менее студентам, уже имеющим некоторый опыт в программировании и желающим глубже понять «железо», рекомендуется самостоятельно ознакомиться с этой книгой целиком.

И наконец, «Основы компьютерной электроники» (авторы и точное название в лекции не указаны) – ещё одна книга из списка, предназначенная для более продвинутых читателей, интересующихся, как компьютер работает на физическом, электрическом уровне. В ней много материала по схемам, электронике, радиотехнике применительно к построению компьютеров. Подобная литература будет полезна будущим специалистам по встроенным системам, разработчикам аппаратуры и тем, кто хочет досконально разобраться, как из транзисторов и прочих электронных компонентов можно создать функционирующую вычислительную машину.

Все упомянутые книги, а также ряд других полезных материалов, собраны преподавателем в электронную библиотеку на Google Диске. Ссылка (QR-код) на эту библиотеку была показана на слайде. Студентам предлагается скачать оттуда интересующие их книги. В папке «архитектура компьютера» находятся, в частности, две версии книги Харриса (для разных типов процессоров) и книга Петцольда (на русском языке). Также там выделены разделы «Информатика» и «Кибернетика», где помещены дополнительные материалы: например, упоминалась книга 1968 года из истории кибернетики (автором, вероятно, является один из пионеров теории нейронных сетей), в которой описан первый перцептрон – эта работа заложила основы нейросетевых моделей. Некоторые части лекционного курса, по словам преподавателя, опираются и на информацию из этой книги. Таким образом, электронная библиотека представляет собой личную коллекцию преподавателя, которую он постоянно пополняет актуальными и интересными на его взгляд изданиями.

Помимо книг, учебные материалы курса включают записи лекций и методические указания к лабораторным работам. Преподаватель сообщил, что все основные лекции записываются и выкладываются в сообществе «Математика + Кибернетика» в социальной сети «ВКонтакте». Это закрытая группа, куда входят студенты текущего и прошлых годов. Новый студент может подать заявку (по предоставленному QR-коду или поиском названия группы) и получить доступ к содержимому. В группе выложены видеозаписи лекций предыдущих лет, которые, хотя могут немного отличаться по форме изложения и вопросам аудитории, в целом соответствуют текущему курсу и содержат все необходимые разъяснения. Также в сообществе публикуются методические материалы, дополнительная литература, посты по тематике кибернетики и смежных областей. Студенты могут сами оставлять сообщения, однако с условием соблюдения тематики сообщества (никаких посторонних обсуждений, только относящееся к курсу математики и кибернетики). Таким образом, группа «Математика + Кибернетика» служит основным информационным центром курса, где можно найти ответы на многие вопросы о заданиях, ознакомиться с опытом студентов прошлых потоков и скачать нужные файлы.

Кроме того, преподаватель разместил записи лекций на внешнем видеопортале (который в настоящее время затруднён к доступу из России). Специально для студентов, находящихся за рубежом или желающих воспользоваться альтернативным ресурсом, был показан QR-код со ссылкой на этот портал. Таким образом, при определённой технической возможности можно посмотреть лекции и на сторонней платформе. Впрочем, основные материалы полностью доступны через ВКонтакте и Google Диск, поэтому у всех студентов есть доступ ко всей необходимой информации.

Современные ИТ-области и примеры кибернетики

После исторического экскурса в кибернетику (на лекции демонстрировались архивные фотографии: первый «компьютерный баг» – насекомое, попавшее в релейный вычислитель и задокументированное в журнале инженеров в 1940-х, эволюция носителей данных – от перфокарт до современных накопителей, внешний вид процессора под микроскопом и т.д.), была затронута тема актуальных тенденций в информационных технологиях. Кибернетика сегодня – обширная область, лежащая в основе многих современных направлений ИТ.

Среди современных направлений ИТ, упомянутых в лекции, – Big Data, искусственный интеллект, Интернет вещей (IoT), квантовые вычисления, технологии виртуальной и дополненной реальности, а также биоинформатика. Big Data связано с анализом огромных массивов данных и поиском в них закономерностей; Интернет вещей подразумевает объединение множества датчиков и устройств в сеть, позволяющую собирать данные и управлять устройствами удалённо; квантовые вычисления – перспективная область на стыке информатики и квантовой физики, разрабатывающая новые типы компьютеров с потенциально очень высокой производительностью; виртуальная и дополненная реальность (VR/AR) открывают новые способы взаимодействия человека с цифровым миром; биоинформатика – междисциплинарная отрасль, применяющая методы ИТ для хранения и анализа биологических данных.

Приводился пример из этой области: современные технологии синтеза ДНК уже позволяют сохранять цифровую информацию на биологических носителях – исследователи уместили несколько петабайт данных в образце размером с обычную флешку. Это наглядно демонстрирует, как идеи, позаимствованные у природы, воплощаются в новых способах хранения данных.

Отдельно подчёркивалось, что достижения кибернетики и информатики пронизывают все сферы жизни – от бытовой техники до космических полётов. Всё разнообразие современных технологий основано на общих принципах обработки информации и управления, о которых идёт речь в курсе. Будущим специалистам важно представлять себе широкий контекст применения своих знаний, понимать, куда движется наука и промышленность. Даже если кто-то собирается заниматься прикладным программированием (разработкой веб-сервисов, игр или мобильных приложений), ему полезно знать об основах кибернетики, поскольку принципы обратной связи, алгоритмы и системы управления лежат в основе любой сложной программы или IT-системы. Кроме того, значительная часть работы программистов сегодня связана с промышленными и корпоративными системами – так называемым enterprise-сектором. Различные системы автоматизации предприятий, учёта, промышленные контроллеры, интеллектуальные системы в производстве – всё это развивается очень активно, и знания кибернетики могут найти применение непосредственно в этих областях.

Навыки и роли современного программиста

Лекция переходит к обсуждению того, каким должен быть современный программист и какие знания требуются в этой профессии. Прозвучало классическое определение: программист – это специалист, который на специализированном языке пишет инструкции для компьютера, чтобы тот выполнил поставленные задачи. Проще говоря, программист переводит задачу (описанную на человеческом языке, например в техническом задании) на язык, понятный машине – то есть на код, который может быть выполнен вычислителем. В этом смысле его можно сравнить с техническим писателем, только пишущим не для людей, а для электронного «исполнителя».

Однако требования к программистам сегодня выходят далеко за рамки умения кодировать. Важно не только знать синтаксис языков программирования, но и уметь проектировать алгоритмы и структуры данных, понимать, как организовать решение задачи наиболее эффективно. Специалист, который владеет лишь конкретным языком программирования и способен следовать готовым алгоритмам, остается, по сути, на уровне начинающего – джуниора. Чтобы расти дальше, необходимо развивать алгоритмическое мышление – умение самостоятельно разбивать сложную проблему на этапы, находить методы её решения и только затем реализовывать в коде. Таким образом, знание и умение применять ключевые алгоритмы (например, сортировки, поиска, работы с базами данных) и структуры данных (списки, деревья, хэш-таблицы и т.д.) отличает более подготовленного специалиста (миддла и выше). Исторически в университетских программах на ИТ-специальностях значительное внимание уделялось именно этим базовым алгоритмам и структурам – их изучают не один семестр, постепенно усложняя задачи. Наша дисциплина также нацелена на то, чтобы дать представление о том, как работает вычислительная техника, на чём основаны алгоритмы и какие базовые возможности предоставляет аппаратное обеспечение для реализации программ.

Кроме алгоритмов, важным компонентом подготовки является понимание принципов работы компьютера на аппаратном уровне. В конце концов, программы выполняются железом, и знание основ вычислительной техники (как устроен процессор, память, какие бывают типы вычислительных устройств) помогает писать более эффективный и правильный код. Компьютер, к которому мы привыкли (настольный ПК, ноутбук, смартфон), – это лишь одна из форм вычислительной машины. Существуют и другие виды: микроконтроллеры во встроенных системах, параллельные вычислительные системы, облачные серверы и т.п. Поэтому хороший программист должен иметь представление, что происходит «под капотом» при исполнении его программ.

Также стоит упомянуть базы данных: практически ни одна серьёзная система не обходится без хранения и обработки данных, поэтому умение работать с системами управления базами данных (SQL и др.), понимание принципов организации данных – обязательная часть квалификации программиста.

В командной разработке к программисту предъявляются и дополнительные требования. Современное программное обеспечение – особенно приложения (applications), то есть крупные программные продукты с пользовательским интерфейсом, сетевыми взаимодействиями, базами данных – создаются командами, а не отдельными людьми. Над одним проектом трудятся разные специалисты: фронтенд-разработчики (отвечающие за пользовательский интерфейс, «лицевую» часть приложения, например, веб-страницы или мобильный интерфейс), бэкенд-разработчики (создающие серверную, «невидимую» пользователю логику приложения – обработку данных, бизнес-логику, взаимодействие с базой данных), специалисты по инфраструктуре и развёртыванию – DevOps-инженеры, тестировщики, а если проект включает элементы ИИ – то и ML-инженеры (разрабатывающие и интегрирующие модели машинного обучения). Всем этим коллективом руководят тимлиды и архитекторы, которые принимают ключевые технические решения, и менеджеры проектов, отвечающие за сроки и связь с заказчиком. Как видно, единичный программист в крупном проекте выполняет лишь часть работы.

Конечно, существуют и фуллстек-разработчики, совмещающие навыки фронтенда и бэкенда, или одиночки, способные создать приложение целиком (особенно если проект небольшой). Например, энтузиаст может в одиночку написать программу или игру – всё зависит от масштаба задачи и уровня мастерства. Однако в индустрии распределение ролей – обычная практика: сложность современного ПО такова, что над ним трудится коллектив, своеобразный «завод по производству программного продукта». Поэтому от выпускников ожидают умения работать в команде и понимать всю технологическую цепочку, хотя бы в общих чертах.

Работа в команде требует от программиста развитых soft skills – навыков коммуникации и сотрудничества. Это не менее важно, чем технические умения. Следует уметь правильно ставить вопросы коллегам, ясно излагать свои мысли, соблюдать деловой этикет (например, договориться, в какой форме общения – на «ты» или на «вы» – принято в коллективе, в каком порядке проходят обсуждения, как презентовать результаты своей работы и т.д.). Важным навыком является ведение документации: любой программный проект сопровождается описаниями, инструкциями, wiki-страницами (например, в том же Confluence или аналогичных системах). Молодой специалист должен понимать, что код – это не всё: нужно также документировать свои решения, писать комментарии, отчёты, чтобы другие члены команды могли быстро разобраться и продолжить работу. Частично этим навыкам учит сама университетская среда: оформление лабораторных отчётов, командные задания – всё это готовит к реальным условиям.

Кроме непосредственных человеческих коммуникаций, софт-скилы программиста включают владение инструментами совместной разработки. Сегодня стандартом индустрии является использование систем контроля версий и управления проектами. Примеры – GitHub или аналогичные платформы для совместного хранения кода, где программисты обмениваются изменениями, ведут историю правок; системы отслеживания задач, такие как Jira (электронная доска задач, помогающая распределять работу внутри команды); средства совместного документирования (типа Confluence), мессенджеры и другие сервисы для координации действий. Освоение этих инструментов – часть профессии. Каждая компания может вводить свои стандарты и наборы технологий для командной работы, но базовые принципы везде схожи. Таким образом, «система коммуникации» в разработке – это совокупность технологий и правил, обеспечивающих эффективное взаимодействие людей при создании ПО.

Преподаватель отмечает, что именно наличие или отсутствие soft skills зачастую отличает успешных специалистов. Этому сложно научить в рамках учебной программы – такие навыки приходят с опытом, через преодоление ошибок в реальных коммуникациях. Тем не менее, студенты должны отдавать себе отчёт, что техническая грамотность должна дополняться умением работать сообща.

Наконец, было обсуждено различие терминов «программа» и «приложение». На английском это program vs application. По сути, программа – это любой самостоятельный фрагмент кода, выполняющий какую-то функцию, тогда как приложением обычно называют законченное программное решение с интерфейсом, взаимодействующее с пользователем и, как правило, состоящее из множества модулей. Приложение можно представить как систему программ. К примеру, небольшая утилита, запускаемая из командной строки для расчёта чего-то – это программа. А сложный программный продукт (например, текстовый редактор, браузер или мобильное приложение) – это уже приложение, т.е. комплекс из множества программных компонентов, работающих вместе и предоставляющих сервис пользователю. Сейчас подавляющее большинство задач требует создания именно приложений, над которыми, как было сказано, работает команда.

Использование нейросетей в обучении программированию

Отдельная важная тема – влияние современных языковых моделей и нейросетевых ассистентов (таких как ChatGPT и аналогичные системы) на процесс обучения программированию. Эти модели способны генерировать код по описанию задачи, предлагать решения, объяснять концепции. Преподаватель отмечает, что языковые модели – мощный инструмент, и те специалисты (во всех областях, не только в IT), которые уже научились ими пользоваться, получают большое преимущество по продуктивности по сравнению с теми, кто их игнорирует. Это объективный факт технологического прогресса. Поэтому студентов призывают не бояться использовать нейросети в учебе и работе.

Однако существует тонкий момент: чрезмерное и неуместное использование таких подсказчиков может затормозить развитие собственных навыков. Если полагаться на ИИ в тех ситуациях, где нужно самому «прокачивать мышцы», мозг недополучает тренировку. В результате может сформироваться зависимость от подсказок, а фундаментальное понимание будет страдать.

Приводилась такая аналогия: языковую модель можно использовать как ускоряющие сапоги, которые позволяют быстрее достигнуть результата, но если опираться на неё постоянно даже в мелочах, она превращается в костыли, без которых человек уже не может выполнить и простейшую задачу. Например, если кто-то перестанет ходить пешком и всё время будет ездить на инвалидном кресле с мотором, его мышцы атрофируются – он разучится ходить. Так же и с интеллектом: если не упражнять собственное мышление в решении задач, а перекладывать всё на нейросеть, то постепенно теряется способность решать даже простые задачи без посторонней помощи.

Как же предложено использовать нейросети разумно? Разделяются две цели обучения:

Получение новой информации, структурирование знаний. Здесь ИИ – отличный помощник. Нейросеть можно спросить о кратком обзоре технологии, о планировании решения, о пояснении нового материала – она быстро выдаст ответ, сэкономив время на поиске по книгам. Для систематизации, повторения, расширения кругозора – эти инструменты незаменимы.

Тренировка навыков, приобретение опыта. А вот здесь важно работать самому. Когда задача заключается в том, чтобы научиться чему-то (например, освоить написание алгоритма сортировки, решить задачу программирования, отладить код), то пусть даже ИИ может мгновенно выдать готовое решение, студент должен сначала попытаться выполнить работу самостоятельно. Только через преодоление сложностей и самостоятельное написание кода приходит понимание и формируются нейронные связи в мозгу, отвечающие за профессиональное мышление.

Применительно к нашему курсу: если цель – узнать что-то новое, понять теорию, можно смело спрашивать у ChatGPT разъяснений, дополнительных примеров. Но если стоит задание написать фрагмент кода или спроектировать устройство, то вместо того, чтобы просто запросить готовый ответ у нейросети, следует попробовать сделать это самому, а уже потом, при необходимости, свериться или улучшить с помощью ИИ. Только в таком режиме нейросеть станет именно «ускоряющими сапогами», а не «костылями».

Преподаватель подчёркивает: нейросети не заменят полностью людей, они заменят тех, кто не умеет ими пользоваться. То есть специалист, эффективно применяющий новые инструменты, останется востребованным, а тот, кто игнорирует прогресс или, напротив, бездумно полагается на него во всём, рискует потерять квалификацию. Программирование – это в первую очередь интеллектуальный труд, требующий понимания задач и умения создавать новые решения. Современные модели способны сильно облегчить рутинные части работы (например, сгенерировать черновой вариант функции, подобрать оптимальный алгоритм из известных), но при решении сложных комплексных проблем всё ещё необходим человек. Машинный интеллект пока не может сам спроектировать большую архитектуру, распределить роли между модулями, оптимизировать решение под все ограничения, понять глубинные потребности пользователя. Тем более не может он придумать принципиально новый алгоритм для совсем новой задачи – он оперирует заложенными в него данными. Поэтому роль программиста смещается в сторону постановки задач и интеграции решений. Даже если будущий инструментарий сократит объём ручного кодирования, прочные знания алгоритмов, структур данных, языков программирования, а также опыт решения задач останутся фундаментом, без которого человек просто не поймёт, что ему предлагает ИИ и как этим правильно воспользоваться.

В лекции был приведён показательный пример: при изучении курса «Алгоритмы и структуры данных» студент теоретически может с помощью нейросети сразу получить готовый код (скажем, функции быстрой сортировки) и даже запросить у неё комментарии к каждой строке, чтобы потом выдать за своё понимание. Но тогда он обманет сам себя – ведь экзаменатор легко может задать уточняющий вопрос, на который, не разобравшись, студент не ответит. Более того, не набив руку на написании простых алгоритмов, невозможно потом сделать что-то сложное: пропустив фазу обучения и практики, нельзя «перепрыгнуть» сразу к экспертному уровню. В итоге тот, кто ленился и списывал у нейросети, окажется неспособен справиться со стоящей перед ним нестандартной задачей, тогда как дисциплинированный студент, научившийся мыслить, сумеет и нейросеть правильно спросить, и свой мозг применить, и результат проверит и доведёт до ума.

Таким образом, рекомендации таковы: активно используйте современные инструменты для расширения знаний, но при выполнении практических заданий старайтесь рассчитывать на себя. Нейросеть – замечательный помощник в роли справочника и репетитора, но при отработке навыка необходимо выполнить работу своими руками. Только сочетая эти подходы, можно максимально эффективно учиться в эпоху ИИ.

Типы программирования: прикладное и системное

Разнообразие сфер программирования означает, что под общим названием «программист» могут скрываться специалисты с разными наборами знаний. На лекции обсуждались основные направления.

Клиент-серверное (прикладное) программирование. Сюда относится разработка приложений, о которой шла речь – фронтенд, бэкенд, мобильные приложения, веб-сервисы. В этой области от программиста требуют знания языков высокого уровня (JavaScript/TypeScript для фронтенда, Python/Java/C#/PHP и т.д. для бэкенда), фреймворков, библиотек, принципов UI/UX и многого другого. Глубоких математических знаний обычно не требуется – акцент делается на умение строить логику приложения, работать с данными, обеспечивать надёжность и масштабируемость.

Системное программирование. Это более «низкоуровневое» направление, включающее разработку операционных систем, драйверов устройств, компиляторов, встроенного (embedded) ПО для микроконтроллеров. Здесь программисту необходимо хорошо знать устройство компьютера, архитектуру процессора, работу памяти, взаимодействие с аппаратурой. Часто используется язык С или даже ассемблер, то есть идёт прямое управление ресурсами машины. Для embedded-разработчика обязательны знания электроники, умение читать схемы, учитывать физические процессы (электрические сигналы, радиосвязь и проч.). Например, тот, кто программирует контроллер для беспилотника (вспомним пример с дроном), по сути занимается системным/встроенным программированием. Но одного контроллера мало – чтобы получился законченный продукт, нужен ещё «обычный» код: интерфейс управления дроном на ПК или смартфоне, сервер для хранения данных полёта и т.д. Этим будут заниматься другие программисты – из области приложений. Таким образом, системное и прикладное программирование часто работают в связке.

Научные вычисления и машинное обучение. Ещё одно большое направление – разработка алгоритмов, опирающихся на математические модели. Сюда относится собственно машинное обучение (ML), о котором мы говорили как о части кибернетики: создание и обучение нейронных сетей, статистический анализ данных, задачи компьютерного зрения, обработки естественного языка и прочие интеллектуальные задачи. Для успешной работы в этой сфере программисту нужно обладать фундаментальной математической подготовкой – разбираться в линейной алгебре, математическом анализе, теории вероятностей. Помимо ML, к этому блоку можно отнести и традиционные алгоритмы искусственного интеллекта (не нейросетевые) – например, логические выводы, алгоритмы оптимизации, поисковые алгоритмы, а также классические алгоритмы компьютерного зрения (которые не используют обучение, а базируются на запрограммированных процедурах обработки изображений). Приводился пример: адаптивная автоматическая коробка передач в автомобиле подстраивается под стиль езды водителя и как бы «обучается» – однако внутри нет никаких нейросетей, просто заложены алгоритмы, меняющие параметры переключения передач на основе накопленной статистики. Такие алгоритмы появились ещё в середине XX века, задолго до нынешнего бума нейронных сетей, и тоже относятся к разделу машинного обучения (в широком смысле, как адаптивные алгоритмы).

Нередко программные проекты требуют сочетания всех трёх направлений. Например, возьмём беспилотный автомобиль: есть встроенное ПО для датчиков и механических систем (оно считывает данные лидаров, управляет двигателем, тормозами – это системное/embedded программирование); есть машинное обучение/ИИ для распознавания объектов и прокладывания маршрута (например, нейросеть анализирует сигнал с камеры, чтобы определить, где дорога, а где пешеход); и есть прикладное ПО – интерфейсы, облачные сервисы, приложения для пользователей, которые позволяют наблюдать за машиной, обновлять её ПО, управлять флотом таких автомобилей через интернет. Каждый компонент пишется специалистами своего профиля, но все они должны понимать друг друга и говорить на «одном языке» технически. Общее у всех – фундамент знаний о том, как работают алгоритмы, какие есть структуры данных, как налажена связь между программой и аппаратной частью. Именно эти основы и призван дать курс.

Подготовка в университете старается заложить широкую базу, после чего выпускник сможет выбрать специализацию. Преподаватель рассказывал, что общался с работодателями (например, с разработчиками в VK, одной из крупных IT-компаний) и спрашивал, какие качества они хотят видеть у молодых специалистов. Ответы разных компаний оказались созвучны: им нужны люди с хорошей базовой эрудицией (включая математическую), с умением учиться и обладающие софт-скилами. Конкретным же технологиям, фреймворкам, языкам они готовы научить на месте – важно, чтобы была платформа для этого, своего рода «скелет», на который нарастят прикладные навыки. Поэтому университетский курс делает упор на фундаментальные дисциплины (математика, алгоритмы, архитектура компьютеров), а также даёт опыт самостоятельного решения задач – через те же лабораторные работы.

Математический фундамент ИТ-специалиста

Как отмечалось, для понимания принципов работы современных технологий необходим определённый уровень математической подготовки. Среди важнейших разделов математики, знания которых пригодятся программисту, были названы:

Математический анализ (в частности, дифференциальное и интегральное исчисление). Даже если напрямую программист не решает интегралы каждый день, понимание непрерывных процессов ценно. Особенно это важно в задачах моделирования, оптимизации, работы с большими данными. На лекции упоминалось, что при решении многих задач искусственного интеллекта фактически приходится работать с уравнениями дифференциального типа. Например, прогнозирование движения объектов, экономические модели, аналитика производства – всё это часто сводится к решению дифференциальных уравнений, причём нередко численными методами (то есть при помощи приближённых вычислительных алгоритмов). Поэтому умение понимать постановку таких задач, представлять себе поведение непрерывных функций и знание базовых численных методов – полезный навык.

Линейная алгебра. Векторы и матрицы – краеугольный камень большинства алгоритмов обработки данных. Любая нейросеть оперирует многомерными весами (это матрицы), изображения представляются матрицами пикселей, при программировании графики или физических задач используются преобразования координат в пространстве (матрицы поворота, масштаба и т.п.). Потому знание линейной алгебры – обязательно: как минимум, нужно понимать, что такое матрица, вектор, уметь выполнять над ними операции, разбираться в понятиях базиса, измерений, собственных значений и т.д.

Теория вероятностей и математическая статистика. Это основа для машинного обучения и большинства методов искусственного интеллекта. Алгоритмы обучения строятся на вероятностных моделях, оценке распределений, поиске корреляций. Да и вообще, в анализе данных без статистики никуда: программисту часто приходится интерпретировать результаты экспериментов, понимать, значимо ли отличие, вычислять вероятности событий (например, отказов системы, поток случайных запросов и нагрузка). Теория вероятностей также лежит в основе многих алгоритмов, используемых в компьютерных науках, поэтому её тоже относят к обязательному «минимуму».

Дискретная математика. Здесь мы подходим непосредственно к тому, с чего начинается цифровая информатика. Дискретная математика изучает структуры, состоящие из отдельных (дискретных) элементов: множества, графы, логические выражения, комбинаторику. В отличие от континуальной (непрерывной) математики, куда относятся упомянутые выше матанализ и линейная алгебра, дискретная оперирует вещами типа целых чисел, логических значений – тем, что в компьютере как раз и реализуется на уровне двоичного кода. В нашем курсе основной упор будет сделан именно на разделы дискретной математики: теорию множеств и математическую логику. Как пояснил преподаватель, эти области – стержень, на котором построена теория вычислимости и, следовательно, вся вычислительная техника.

Отдельно упоминались еще теория типов и другие разделы современной математики, однако они выходят за рамки необходимого минимума и интересны скорее энтузиастам. Была подчеркнута роль теории множеств и математической логики как основных «китов», на которых основывается вся строгая математика. В конце XIX – начале XX века развитие этих разделов фактически заложило фундамент всей современной науки о вычислениях. Первые аксиомы и понятия теории множеств ввёл немецкий математик Георг Кантор (определив множество как «многие, мысленно объединённые в одно»). На базе его идей затем строилась формализация математики. Математическая логика оформилась чуть позже, в работах Джорджа Буля и др., о чём ниже.

Важно понять: вся компьютерная система – на уровне программ и схем – в конечном счёте сводится к дискретным математическим конструкциям. Лектор привёл впечатляющий тезис: все достижения человеческой мысли – сложнейшие программы, искусственный интеллект, игры, космические корабли – работают на основе всего лишь пяти базовых элементов. Это:

логические значения: истина (True, логическая 1) и ложь (False, логический 0);

логические операции: И (AND), ИЛИ (OR), НЕ (NOT).

Иными словами, любые данные можно закодировать последовательностями бит (0 и 1), а любое преобразование данных – реализовать комбинацией базовых логических операций. Поэтому вполне справедливо сказать, что единицы, нули и примитивные булевы функции лежат в основе всего разнообразия вычислений. Лекция ставит задачу объяснить, как из этих простейших «кирпичиков» шаг за шагом строится вся вычислительная техника и программное обеспечение. Осознать этот процесс – значит понять, как из логики рождается компьютер.

Формальная логика и основы вычислений

В заключительной части лекции даётся введение в математическую (формальную) логику, которая как раз и обеспечивает научный аппарат для работы с понятиями истины и лжи. Исторически логика сначала развивалась как раздел философии: еще Аристотель в IV веке до н.э. в трактате «Органон» сформулировал законы правильного мышления, принципы построения умозаключений (например, закон непротиворечия, правила дедукции и индукции и др.). Эта философская логика описывает то, как люди рассуждают в повседневной речи.

Спустя многие столетия, в XIX веке, английский математик Джордж Буль совершил революцию, предложив математизировать логику. Вместе с другими учёными того времени (Огастус де Морган, Чарльз Пирс, Генри Шеффер) он разработал систему, получившую название булева алгебра – по сути, это и есть современная формальная логика, или математическая логика. Идея Буля состояла в том, чтобы взять высказывания (предложения на человеческом языке) и записать их в виде символов, а операции над суждениями представить как некие алгебраические законы. До того алгебра оперировала только числами, а Буль и коллеги заставили её оперировать понятиями «истинно» и «ложно».

Таким образом, родилась формальная система логики. Что такое формальная система? В общем виде – это строго определённый математический язык и правила оперирования на нём. Формально систему можно задать как кортеж из четырёх множеств:

Алфавит ($V$) – набор базовых символов, которые используются (например, в арифметике это цифры 0–9, знаки операций +, –, =, переменные x, y, скобки и т.д.; в логике – символы для констант True/False, обозначения логических операций, переменные для высказываний, знаки связок).

Правильно построенные формулы ($F$) – множество всех корректных формул, составленных из символов алфавита. Не всякая произвольная комбинация знаков имеет смысл: мы вводим правила грамматики. Например, в арифметике строка «1+1=2» – это правильно построенная формула, а просто «1+1» – нет, потому что пропущен знак сравнения (знак равенства или другой оператор). Точно так же и в логике: определяются правила, как можно составлять высказывания из символов (например, «A И (НЕ B)» – корректная формула, а «ИЛИ А B» – некорректная, так как нарушен синтаксис). Правила построения формул можно задавать явно (перечисляя допустимые комбинации символов) или неявно – через грамматику, то есть рекурсивное определение. К примеру, можно сказать: «любая переменная – это формула; если P – формула, то (НЕ P) – тоже формула; если P и Q – формулы, то (P И Q) – формула; и т.д.». Таким образом задаётся бесконечное множество всех выражений, которые считаются «правильными» в нашей системе.

Аксиомы ($A$) – набор исходных утверждений, принимаемых в данной системе без доказательства. Они задают базовые истины и ограничения системы. Например, в классической арифметике есть аксиома, что деление на 0 не определено (то есть невозможно дать результат операции 1/0). В геометрии Евклида – аксиома, что через две точки проходит прямая и параллельные прямые не пересекаются. Мы можем вообразить другую систему геометрии, в которой параллельные линии пересекаются (на сфере, например) – тогда изменится аксиоматика, но внутри каждой такой системы рассуждения останется непротиворечивость. В логике также можно задавать аксиомы. Например, классическая двузначная логика принимает закон исключённого третьего: любое высказывание либо истинно, либо ложно, третьего не дано. Если эту аксиому убрать или изменить, получим многозначную логику (как, например, в индийской философии существовали представления о том, что истина и ложь не исчерпывают всех возможностей). В целом аксиоматика определяет, в каком «мире» мы работаем – реальном или воображаемом.

Правила вывода ($R$) – совокупность шаблонов логического вывода (инференции). Они описывают, как из одних формул можно получить другие. Иными словами, это логические схемы типа: «если A истинно и верно, что из A следует B, то B тоже истинно». Примером правила вывода служит принцип modus ponens: из посылок «A истинно» и «Если A, то B» выводится заключение «B истинно». Правила вывода обычно записываются схематично (например: A, A→B ⊢ B). В формальной системе правила вывода задаются явно в виде шаблонов, по которым можно из аксиом (и ранее выведенных формул) получать новые формулы. В программировании аналогом служит конструкция if ... then ... – по сути, реализация логического вывода: «если условие выполнено, то выполнить действие».

Определив алфавит, формулы, аксиомы и правила вывода, мы создаём формальную систему. В ней можно выводить теоремы – то есть формулы, полученные из аксиом путём применения последовательности правил. Формальная система декларативна: мы заранее фиксируем все допущения и законы, вместо того чтобы брать их из наблюдаемого мира. Иначе говоря, мы декларируем, что в нашей системе действуют определённые правила (например, понятие «неистина» трактуется как отрицание истины), и руководствуемся ими. Благодаря этому формальные системы не зависят от реального мира – они могут описывать как нашу физическую реальность, так и любой выдуманный мир, главное, чтобы не было внутренних противоречий.

Джордж Буль, таким образом, сформировал формальную систему для логики высказываний. Алфавит у неё включал переменные (обозначающие произвольные утверждения) и логические связки; множество формул определялось рекурсивно по описанным правилам; аксиомами служили базовые законы (например, законы идемпотентности и коммутативности для операций И/ИЛИ, закон двойного отрицания и др.); а правилами вывода – стандартные логические шаги (из A и A→B выводить B и т.д.). Это позволило рассуждать об истинности высказываний не на бытовом языке, а с помощью формальных символов – как при решении уравнений.

Какова практическая ценность логики? Она даёт универсальный каркас для вывода истины. Используя логические правила, человек не узнаёт новых фактов об окружающем мире (их надо получать экспериментом и наблюдением), но он может из уже известных фактов вывести новые следствия. Например, если известно, что «Если идёт дождь, то земля мокрая» и дано «Идёт дождь», то логика подсказывает: «земля мокрая». Это тривиальный пример, но в сложных системах, содержащих множество условий, именно формальная логика позволяет автоматизировать процесс получения верных выводов.

Одно из ключевых понятий логики – высказывание (или суждение). Дают определение: суждение – это любое предложение на человеческом языке, о котором можно однозначно сказать, что оно либо истинно, либо ложно. Иными словами, суждение – это утверждение, обладающее определённым логическим значением. Примеры: «Жить хорошо» – это суждение (можно обсуждать, истинно оно или ложно, но оно что-то утверждает); «2+2=4» – тоже суждение (конкретно истинное в принятой системе аксиом стандартной арифметики). А вот просто слово «ботинки» – не суждение, потому что само по себе оно ничего не утверждает и ни с чем не соотносится. Не являются суждениями вопрос («Что такое кибернетика?») или побуждение («Встаньте, пожалуйста») – у них иной смысл, они не несут категории истинности. Даже предложение с неопределёнными компонентами, например «X + 1 = 0», не будет суждением, пока не конкретизировать переменную: для одних X оно истинно, для других ложно, но само по себе, без контекста, оно не имеет определённого значения истины. Чтобы такая фраза стала полноценным суждением, надо её либо уточнить («1+1=0» – заведомо ложное суждение, «-1+1=0» – истинное), либо рассматривать как схему в рамках формальной системы (например, утверждение «существует X, для которого X+1=0» – это уже суждение, и в обычной арифметике оно истинно, поскольку таким X является -1).

Логика оперирует высказываниями, комбинируя их с помощью логических связок («и», «или», «не», «если ..., то ...», «либо ..., либо ...» и т.д.). Например, из двух суждений A и B можно составить сложное: «A И B» (оно истинно, только если оба исходных высказывания истинны), «A ИЛИ B» (истинно, если истинно хотя бы одно из них) и т.п. Отдельно отмечается связка «исключающее или» (XOR), когда из двух утверждений истинно строго одно – такая операция также часто используется и будет рассмотрена по курсу. Также упоминается штрих Шеффера (операция NAND) – уникальная логическая функция, с помощью которой, как известно, можно выразить все остальные (то есть NAND образует функционально полную систему). Все эти логические операции и законы (включая законы де Моргана, которые связывают И/ИЛИ с НЕ) будут подробно рассмотрены на практических занятиях по булевой алгебре.

Был задан вопрос: может ли существовать предложение, которое по форме звучит как суждение («о нём можно подумать: истина или ложь?»), но на самом деле таковым не является. Студенты вспомнили так называемые логические парадоксы – например, знаменитое утверждение «Это предложение ложно». Попытка определить его истинность приводит к противоречию (если предположить, что оно истинно, получается, что оно ложно, и наоборот). Другой парадокс – вопрос о всемогуществе: «Может ли всемогущий Бог создать камень, который сам не сможет поднять?» В логике подобные вещи считаются некорректными висказиваниями, не имеющими определённого значения истины.

Анализ таких парадоксов привёл к глубоким результатам в математике. Так, английский философ и математик Бертран Рассел выявил парадокс теории множеств о «бродобреe», который бреет всех, кто не бреется сам (вопрос: кто бреет самого брадобрея?). Эти проблемы поставили под сомнение надежность «наивной» теории множеств. Австрийский логик Курт Гёдель в 1931 году доказал свою знаменитую теорему о неполноте: в любой достаточно мощной формальной аксиоматической системе существуют истинные утверждения, которые не выводимы (не доказуемы) в рамках этой системы. Проще говоря, невозможно, оставаясь внутри системы, доказать все истины о ней самой – всегда найдётся утверждение, которое нельзя ни доказать, ни опровергнуть средствами системы (например, утверждение «Я недоказуем» относительно данной системы). Для доказательства непротиворечивости любой такой системы требуется выйти за её пределы. Гёдель тем самым показал пределы возможностей формализованного подхода.

В контексте нашего курса достаточно понимать, что формальная логика, лежащая в основе вычислительной техники, работает в рамках двоичных (булевых) моделей, где подобных парадоксов не возникает – компьютер не пытается доказать собственную непротиворечивость, он просто выполняет заложенные в него алгоритмы. Нам важно уяснить базовые принципы формальных систем и логики, чтобы затем увидеть, как на них строятся элементы компьютерных схем и программ. Эти вопросы будут углублённо рассмотрены на следующих лекциях. Таким образом, вводная лекция заложила философско-математический фундамент курса, познакомив с организационными моментами и ключевыми понятиями кибернетики, без которых невозможно двигаться дальше к пониманию устройства вычислительных машин.