Д.Г. ПОДОБЕД,
М.В. ПОДОБЕД,
О.В. ПОДОБЕД

ОСНОВЫ ИНФОРМАТИКИ

(базовые материалы для курса лекций)

Учебное пособие

Санкт-Петербург

2010

Министерство образования и науки
Российской федерации

Государственное образовательное учреждение
высшего профессионального образования

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ
РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ

Д.Г. ПОДОБЕД, М.В. ПОДОБЕД, О.В. ПОДОБЕД

ОСНОВЫ ИНФОРМАТИКИ

(базовые материалы для курса лекций)

Учебное пособие

Санкт-Петербург
2010

ББК 22.18я7
П 444
УДК 681.3:65.5

Подобед Д.Г., Подобед М.В., Подобед О.В. Основы информатики (базовые материалы курса для лекций): учебное пособие / ГОУВПО СПбГТУРП. - СПб., 2010.- 75 с.

В учебном пособии изложены основные определения базовых понятий по тематике, которые необходимы при изучении материалов и тестировании студентов по курсу «Информатика».

Предназначается для организации работы студентов, обучающихся по специальностям: «Экономика и управление на предприятии ЦБП», «Менеджмент организации».

Рецензенты:
доцент кафедры прикладной математики и информатики ГОУВПО СПбГТУРП Антонюк П.Е.;
доцент кафедры эконометрии ГОУВПО «Санкт-Петербургский государственный университет водных коммуникаций», канд. техн. наук Полянская Т.И.

Рекомендовано к изданию Редакционно-издательским советом университета в качестве учебного пособия.

ТЕМА 1. БАЗОВЫЕ ОПРЕДЕЛЕНИЯ

Требования: студенты должны достаточно точно формулировать определения основных понятий курса «Информатика», знать исторические корни дисциплины, базовые формулы, должны свободно оперировать различными единицами измерения объема информации, переходить от одних единиц к другим, представлять себе сравнительные объемы информации, системы компьютерного двоичного кодирования.

Информация (от лат. informatio – осведомление, разъяснение, изложение) — в широком смысле абстрактное понятие, имеющее множество значений, в зависимости от контекста. В узком смысле этого слова под «информацией» понимают сведения (сообщения, данные) независимо от формы их представления.

Понятие «информация» рассматривалось еще античными философами. До начала промышленной революции определение этого понятия оставалось прерогативой, преимущественно, философов.

Понятие «информация» в физике определяется следующим образом. Объекты материального мира находятся в состоянии непрерывного изменения, которое сопровождается обменом энергии. При этом изменение состояния одного объекта приводит к изменению состояния другого объекта. Это явление, вне зависимости от того, как, какие именно состояния и каких именно объектов изменились, называется передачей ***сигнала*** от одного объекта другому. Изменение состояния объекта при передаче ему сигнала называется регистрацией сигнала. Сигнал или последовательность сигналов образуют сообщение, которое может быть воспринято получателем в том или ином виде, а также в том или ином объёме. Информация есть термин, качественно обобщающий понятия «сигнал» и «сообщение». Если сигналы и сообщения можно исчислять количественно, то применительно к информации можно сказать, что сигналы и сообщения являются единицами измерения объёма информации.

Далее вопросы теории информации стала рассматривать новая на то время наука ***кибернетика***. В статистической теории передачи информации не раскрывается то, что следует обозначать как «информация». Клод Шеннон подразумевает под термином «информация» нечто фундаментальное (нередуцируемое), то есть категорию. Интуитивно полагается, что информация имеет содержание. ***Информация уменьшает общую неопределенность и доступна измерению***.

Основоположник кибернетики Норберт Винер писал, что информация есть информация, а не материя и не энергия. Тот материализм, который не признает этого, не может быть жизнеспособным в настоящее время... Если в ходе взаимодействия между объектами один объект передаёт

другому некоторую субстанцию, но при этом сам её не теряет, то эта субстанция называется информацией, а взаимодействие - информационным.

Информация «многолика», включает в себя синтаксический, семантический и прагматический аспекты. ... Разные стороны понятия информации отображаются в целом спектре теорий. Эти теории, как правило, не противоречат, а дополняют друг друга, развивая разные количественные меры, связанные с той или иной стороной феномена информации. При этом всегда имеется в виду задача - если не полного, то частичного - синтеза этих теорий.

Общим в определениях термина «информация» являются различные формы регистрации объективных событий окружающего мира и субъективный механизм анализа таких событий.

Отсюда следует, что ***информация – совокупность объективных данных и субъективных методов их обработки***.

Под качеством информации понимают степень её соответствия требованиям потребителей.

Свойства информации являются относительными, так как зависят от потребностей потребителя информации.

Выделяют следующие свойства, характеризующие качество информации:

- ***объективность информации*** характеризует её независимость от чьего-либо мнения или сознания, а также от методов получения; более объективна та информация, в которую методы получения и обработки вносят меньший элемент субъективности;
- ***полнота***: информацию можно считать полной, когда она содержит минимальный, но достаточный для принятия правильного решения набор показателей; как неполная, так и избыточная информация снижает эффективность принимаемых на основании этой информации решений;
- ***достоверность*** - свойство информации быть правильно воспринятой.

Объективная информация всегда достоверна, но достоверная информация может быть как объективной, так и субъективной. Причинами недостоверности могут быть: преднамеренное искажение (дезинформация), непреднамеренное искажение субъективного свойства, искажение в результате воздействия помех, ошибки фиксации информации.

В общем случае достоверность информации достигается: указанием времени свершения событий, сведения о которых передаются;

сопоставлением данных, полученных из различных источников; своевременным вскрытием дезинформации; исключением искаженной информации и др.

Адекватность информации — степень соответствия реальному объективному состоянию дела.

Доступность информации — мера возможности получить ту или иную информацию.

Актуальность информации — степень соответствия информации текущему моменту времени.

Также ***можно классифицировать свойства информации, характеризующие её качество***:

- ***содержательность,*** или внутреннее качество (качество, присущее собственно информации и сохраняющееся при её переносе из одной системы в другую);
- ***значимость*** (свойство сохранять ценность для потребителя с течением времени);
- ***полнота*** (свойство, характеризуемое мерой её достаточности для решения определенных задач);
- ***идентичность*** (свойство, заключающееся в соответствии информации состоянию объекта);
- ***кумулятивность*** (свойство информации, заключённой в массиве небольшого объёма, достаточно полно отражать действительность);
- ***избирательность***;
- ***гомоморфизм*** ;
- ***сохранность;***
- ***конфиденциальность***.

Для человека информация подразделяется на виды в зависимости от типа воспринимающих её рецепторов:

- ***визуальная*** - воспринимаемая органами зрения;
- ***аудио*** - воспринимаемая органами слуха;
- ***тактильная*** - воспринимаемая тактильными рецепторами;
- ***обонятельная*** - воспринимаемая обонятельными рецепторами;
- ***вкусовая*** - воспринимаемая вкусовыми рецепторами.

По форме представления информация делится на следующие виды:

- ***текстовая*** - передаваемая в виде символов, предназначенных обозначать лексемы языка;
- ***числовая*** - в виде цифр и знаков, обозначающих математические действия;
- ***графическая -*** в виде изображений (событий, предметов, графиков);
- ***звуковая*** - устная или в виде записи передача лексем языка аудиальным путем.

По предназначению информация делится на следующие виды:

- ***массовая*** - содержит тривиальные сведения и оперирует набором сведений, понятным большей части социума;
- ***специальная*** - содержит специфический набор понятий, при использовании происходит передача сведений, которые могут быть не понятны основной массе социума, но необходимы и понятны в рамках узкой социальной группы, где используется данная информация;
- ***личная*** - набор сведений о какой-либо личности, определяющий социальное положение и типы социальных взаимодействий внутри популяции.

Дезинформацией (также дезинформированием) называется один из способов манипулирования информацией, как то: введение кого-либо в заблуждение путем предоставления неполной информации, искажения контекста, искажения части информации.

С юридической точки зрения, информация - это сведения о людях, предметах, фактах, событиях и процессах, независимо от формы их представления. Данное определение зафиксировано в Законе «Об информации, информатизации и защите информации», утвержденном в 1995 году.

Согласно принятому законодательству, информация признана объектом гражданских прав с определением норм и правил отнесения ее к массовой, персональной или конфиденциальной.

Особую роль для общества играет документированная информация. Документы - это информация, зафиксированная на материальном носителе - бумаге или машинном носителе, имеющем реквизиты, позволяющие его идентифицировать.

С понятием «информация» связаны две научные и учебные дисциплины: «Информатика» и «Теория информации».

«Информатика» - от франц. - объединение слов «***информа***ция» и

«автома***тика***» - наука, изучающая все аспекты получения, хранения, преобразования, передачи и использования информации. Развитие информатики тесно связано с появлением ЭВМ, поэтому на сегодняшний день информатика - техническая наука, систематизирующая приемы создания, хранения, воспроизведения, обработки и передачи данных средствами вычислительной техники, а также наука о принципах функционирования этих средств и методах управления ими. Эта наука сформировалась сравнительно недавно, и потому нет той строгости и четкости в определениях, которые присущи математике, физике и т.д.

Информатика - это новая научная дисциплина и новая информационная индустрия, связанные с использованием персональных компьютеров и сетей ЭВМ. В новом тысячелетии предполагается, что основная информация, связанная с деятельностью людей, будет храниться в памяти электронных вычислительных машин. Развитие бизнеса, образования, промышленности и общества в целом учеными, политиками, бизнесменами во многом связывается с широким использованием информационных ресурсов Интернета и нарастающими интеллектуальными возможностями вычислительных машин.

Информатика как научная дисциплина изучает законы, принципы и методы накопления, обработки и передачи информации с помощью ЭВМ. В этом смысле информатика как наука является фундаментом для развития новой информационной индустрии, основанной на использовании сетей ЭВМ.

Фундамент информатики образуют вычислительные науки - науки об вычислительных процессах и организации вычислительных машин, вычислительных систем и сетей. Основным объектом вычислительных наук являются вычислительные машины - устройства для организации вычислений и обработки символьной информации.

Обработка, накопление и передача информации происходит не только внутри ЭВМ. Передачу и накопление информации мы видим при общении людей, в технических устройствах, в живых организмах и в жизни общества, что тоже входит в предмет изучения информатики как научной дисциплины.

Передача информации в общении людей - это передача сведений и суждений, данных и сообщений. Даже улыбка является передачей информации при общении людей друг с другом. Любая совместная деятельность людей - работа, учеба и даже игра - построены на обмене и передаче информации.

Для живых существ восприятие и передача информации в форме сигналов - основное отличие от неодушевленных предметов окружающего мира. Речевая форма передачи знаковой информации - основное отличие

людей от других живых существ.

Точкой отсчета становления информатики как индустрии стало изобретение в середине XX века ***электронных вычислительных машин.*** Основной особенностью компьютеров стала возможность автоматической обработки информации. Переработка информации перестала быть исключительной способностью людей и живых существ.

Параллельно в середине XX века были заложены теоретические основы информатики как научной дисциплины. В этот период получили развитие математическая логика - фундамент теоретической информатики и теория алгоритмов - фундамент вычислительных наук.

Целью вузовского курса «Информатика» считается освоение профессионального использования персональных компьютеров и решения на ЭВМ профессиональных задач. Для развития этих умений необходима определенная культура и развитие логического мышления.

Эффективное использование ЭВМ предполагает наличие ***информационной культуры*** - умения искать, передавать и подготавливать информацию в форме текстов и рисунков с помощью персональных компьютеров и сети Интернет. Развитие этой культуры ведет к более глубокому развитию логического мышления.

Логическое мышление проявляется в умении решать различные интеллектуальные задачи и, в том числе, в решении сложных задач с помощью ЭВМ. Эти интеллектуальные способности выражаются в умениях рассуждать, доказывать, ставить задачи, а также подбирать и обосновывать способы их решения.

Сложность изучения информатики как индустрии связана с ее беспрецедентной динамичностью: технические средства информатики - компьютеры, программы и средства телекоммуникаций полностью модернизируются каждые пять - шесть лет, а соответствующие технические знания обновляются каждые два - три года. Одновременно модернизируются компьютерные сети, архивы, библиотеки и информационные системы.

Информатика как научная дисциплина сохраняет свое ядро - общие принципы, законы и методы организации вычислений и обработки информации с помощью ЭВМ. Эти принципы сохранят свою роль и значение для всех моделей и типов ЭВМ, независимо от их элементной базы, быстродействия и объемов памяти. Более того, общие законы информатики как общие законы интеллектуальной деятельности, сохраняют свою силу при изучении принципов обработки, накопления и передачи информации не только в ЭВМ, но и в живых организмах и человеческом обществе.

Изучение курса «Информатика» предполагает обязательное решение

различного типа задач с использованием современных инструментальных сред, в частности, Excel и Mathcad. Применительно к инженерным специальностям, основное внимание в курсе обращается на решение различных прикладных задач в различных инструментальных средах с целью выработки определенных навыков пользователей персональных компьютеров, умения грамотно формулировать задачи и оценивать полученные с помощью средств вычислительной техники результаты по целому ряду признаков.

Теория информации (математическая теория связи) — раздел прикладной математики, определяющий понятие информации, её свойства и устанавливающий предельные соотношения для систем передачи данных. Как и любая математическая теория, оперирует математическими моделями, а не реальными физическими объектами (источниками и каналами связи). Использует, главным образом, математический аппарат терии вероятностей и математической статистики.

Информация и кодирование — два основных понятия современной информационной техники. Информация в техническом смысле этого слова и методы защиты информации от ошибок, возникающих в результате передачи сообщений, являются сегодня основой при подготовке специалистов, работающих в области информационных технологий.

Основные разделы теории информации— кодирование источника (сжимающее кодирование) и канальное (помехоустойчивое) кодирование. Теория информации тесно связана с криптографией и другими смежными дисциплинами.

Понятие количества информации совершенно естественно связывается с классическим понятием статистической механики – понятием энтропии. Как количество информации в системе есть мера организованности системы, точно так же энтропия системы есть мера дезорганизованности системы: одно равно другому,взятому с обратным знаком.

Возникновение математической теории информации стало возможным после того, как было осознано, что количество информации можно задать числом так же, как можно выразить числом расстояние, время, массу тела, количество тепла и т. д.

Клода Шеннона (1916-2001) называют «отцом теории информации». Его теория изначально понималась как строго математическая задача в статистике и дала инженерам средств передачи информации путь к определению емкости коммуникационного канала в терминах количества бит. Передающая часть теории не занимается значением (семантикой) передаваемого сообщения, однако дополняющая часть теории информации обращает внимание на содержимое через сжатие с потерями субъекта

сообщения, используя критерий точности.

Количество информации определяется формулами Хартли и Шеннона.

В 1928 г. американский инженер Р. Хартли предложил научный подход к оценке сообщений.

Предложенная им формула имела следующий вид:

$$\mathbf{I = \log_2 K},$$

где К - количество равновероятных событий;

I - количество бит в сообщении, такое, что любое из К событий произошло. Тогда

$$\mathbf{K=2^I}.$$

Иногда формулу Хартли записывают так:

$$\mathbf{I = \log_2 K = \log_2 (1 / p) = - \log_2 p},$$

так как каждое из К событий имеет равновероятный исход

$$\mathbf{p = 1 / K, \text{ то } K = 1 / p}.$$

В 1948 г. американский инженер и математик К. Шеннон предложил формулу для вычисления количества информации для событий с различными вероятностями.

Если I - количество информации, К - количество возможных событий, p_i - вероятности отдельных событий, то количество информации для событий с различными вероятностями можно определить по формуле

$$\mathbf{I = - sum[p_i * \log_2 p_i]},$$

где i принимает значения от 1 до К.

Формулу Хартли теперь можно рассматривать как частный случай формулы Шеннона:

$$\mathbf{I = - sum [(1/K) * \log_2 (1/K)] = I = \log_2 K}.$$

При равновероятных событиях получаемое количество информации максимально.

Теория информации описывается с помощью вероятностных диаграмм кодирования и передачи информации в конкретных естественно-научных приложениях. При этом появляется возможность для анализа и оптимизации потоков информации в технических системах.

Под ***кодированием информации*** понимают некоторое отображение сообщения по известным правилам. При этом, в шенноновской модели передачи информации, блоки передатчика и приемника следует расширить в соответствии с этими правилами. Между кодированием источников и кодированием каналов существует четкое различие. Примерами кодирования источников могут служить передача связанного текста кодом Морзе, оцифровка аудиосигнала при записи на компакт-диски. При кодировании источников избыточность сообщений снижается и такое кодирование часто называют сжатием данных. Кодирование каналов, наоборот, увеличивает избыточность сообщений. Внесение дополнительных проверочных символов позволяет обнаруживать и даже исправлять ошибки, возникающие при передаче информации по каналу. Кодирование канала в дальнейшем будем называть ***помехоустойчивым кодированием***. Без помехоустойчивого кодирования было бы невозможным создание накопителей огромной емкости, таких, как CD-ROM, DVD или жестких дисков. Дополнительные затраты на помехоустойчивое кодирование, которое обеспечивает приемлемые вероятности ошибок записи / чтения, становятся пренебрежимо малыми по сравнению с выигрышем от достигаемой при этом плотности записи.

Количество информации в теории информации определяется как мера информации, сообщаемой появлением события с определенной вероятностью; мера оценки информации, содержащейся в сообщении; мера, характеризующая уменьшение неопределенности, содержащейся в одной случайной величине относительно другой.

Объёмы информации можно представлять как логарифм по основанию 2 количества состояний. Наименьшее целое число, логарифм которого положителен, 2. Соответствующая ему единица - ***бит*** является основой исчисления информации в цифровой технике; бит (bit ; от англ.binary - двоичная + digit - цифра).

Бит - минимальная единица измерения количества передаваемой или хранимой информации, соответствующая одному двоичному разряду, способному принимать значений 0 или 1. Единица, соответствующая числу 3, (***трит***) равна $\log_2(3)=1{,}585$ бита, числу 10 (***хартли***) - $\log_2(10)=3{,}322$ бита. Такая единица как ***нат*** (nat, е-бит), соответствующая натуральному логарифму применяется в вычислительной технике в инженерных и научных расчётах.

Машинное слово — машинозависимая и платформозависимая величина, измеряемая в битах или байтах (тритах или трайтах), равная разрядности регистров процессора и/или разрядности шины данных (обычно некоторая степень двойки). На ранних компьютерах размер слова совпадал также с минимальным размером адресуемой информации (разрядностью данных, расположенных по одному адресу); на современных компьютерах

минимальным адресуемым блоком информации обычно является байт, а слово состоит из нескольких байтов.

Машинное слово определяет следующие характеристики аппаратной платформы: разрядность данных, обрабатываемых процессором; разрядность адресуемых данных (разрядность шины данных); максимальное значение беззнакового числа целого типа, напрямую поддерживаемого процессором: если результат арифметической операции превосходит это значение, то происходит переполнение.

При кодировании информации по Y разрядам с помощью X символов количество возможных различных комбинаций N определяется по формуле $N=X^Y$ (это соотношение определяет число размещений с повторениями). При двоичном кодировании (X=2) количество возможных различных комбинаций N определяется по формуле $N=2^Y$.

Напомним таблицы размерностей:

 1 бит - самая маленькая единица информации — условно один «0» или одна «1».
 ***1 байт* = 8 бит (8 = 2^3)**; в международной системе кодов ASCII (Fmtrican Standard Code for Information Interchange, Американский стандартный код обмена информацией) каждый символ кодировался одним байтом, что позволяло закодировать 2^8 = 256 символов, чего на первых порах хватало. Сейчас происходит переход к кодировке Unicode, где каждый символ кодируется двумя байтами, что позволяет кодировать 2^{16} = 65536 символов, многократно увеличивая возможности кодирования.
 ***1 Кбайт (килобайт)* = 1024 байт (2^{10} байт)**. По этому поводу есть анекдот, что физик думает, что в одном килобайте 1000 байт, а программист - что в одном килограмме 1024 грамма.
 ***1 Мбайт (мегабайт)* = 1024 Кбайта (2^{10} Кбайт или 2^{20} байт).**
 ***1 Гбайт (гигабайт)* =1024 Мбайта (2^{10} Мбайт или 2^{30} байт).**
 ***1 Тбайт (терабайт)* =1024 Гбайта (2^{10} Гбайт или 2^{40} байт).**

В недалеком будущем нас ожидают:

 1 Пбайт (петабайт) =1024 Тбайта (2^{10} Тбайт или 2^{50} байт).
 1 Эбайт (экзабайт) =1024 Пбайта (2^{10} Пбайт или 2^{60} байт).
 1 Збайт (зеттабайт) =1024 Эбайта (2^{10} Эбайт или 2^{70} байт).
 1 Йбайт (йоттабайт) =1024 Збайта (2^{10} Збайт или 2^{80} байт).

ТЕМА 2. СИСТЕМЫ СЧИСЛЕНИЯ

Требования: студенты должны знать основные позиционные и непозиционные системы счисления, их свойства, иметь понятие о базисе и основании системы счисления, свободно ориентироваться в десятичной, двоичной, восьмеричной, шестнадцатеричной системах счисления, переводить числа из одной системы в другую (пользуясь, в том числе, и «триадами» и «тетрадами»), осуществлять простейшие арифметические действия над числами в разных системах счисления.

Система счисления:

- даёт представления множества чисел (целых или вещественных);
- даёт каждому числу уникальное представление (или, по крайней мере, стандартное представление);
- отражает алгебраическую и арифметическую структуру чисел.

Например, привычное десятичное представление целых чисел уникальным образом формирует каждое целое число как конечную последовательность цифр.

Арифметические операции (сложение, вычитание, умножение и деление) суть стандартные арифметические алгоритмы. Тем не менее, когда десятичное представление используется для рациональных или вещественных чисел, оно более не уникально. Многие рациональные числа имеют две записи - стандартная десятичная дробь (например, 2.31) и периодическая (например, 2.309999999...). Десятичные дроби не имеют ненулевых цифр после некоторой заданной позиции. Например, числа 2.31 и 2.310 обычно считаются одинаковыми, за исключением экспериментальных наук, где нулевыми младшими разрядами обозначается точность представления.

Системы счисления подразделяются на ***позиционные***, ***непозиционные*** и ***смешанные***.

В ***позиционных системах счисления*** один и тот же ***числовой знак (цифра)*** в записи числа имеет различное значение в зависимости от того места (***разряда***), где он расположен. Изобретение позиционной нумерации, основанной на поместном значении цифр, приписывается шумерам и вавилонянам; развита была такая нумерация индийцами и имела неоценимые последствия в истории человеческой цивилизации. К числу таких систем относится современная десятичная система счисления, возникновение которой связано со счётом на пальцах. В средневековой Европе она появилась через итальянских купцов, в свою очередь, заимствовавших её у мусульман.

Каждая позиционная система счисления определяется некоторым целым числом $b > 1$ (***основание системы счисления***) таким, что b единиц в каждом разряде объединяется в одну единицу следующего по старшинству разряда. Система счисления с основанием b также называется ***b-ричной***.

Целое число x в b-ричной показательной системе счисления

представляется в виде конечной линейной комбинации степеней числа b:

$$\mathbf{x=sum(a_k*b^k;\ k=0,...,n-1)}\ ,$$

где a_k - это целые числа, называемые цифрами, удовлетворяющие неравенству $0 \le a_k < b$.

Каждая степень b^k в такой записи называется *b*-ричным разрядом, старшинство разрядов и соответствующих им цифр определяется значением показателя *k*. Обычно для ненулевого числа *x* требуют, чтобы старшая цифра a_{n-1} в b-ричном представлении *x* была также ненулевой. Если не возникает разночтений (например, когда все цифры представляются в виде уникальных письменных знаков), число *x* записывают в виде последовательности его *b*-ричных цифр, перечисляемых по убыванию старшинства разрядов слева направо: $x = a_{n-1}\ a_{n-2}\ \ldots\ a_0$. Например, число *сто три* представляется в десятичной системе счисления в виде: $103=1*10^2 + 0*10^1 + 3*10^0$.

Наиболее употребимыми в настоящее время позиционными системами являются:

- ***единичная система счисления*** (как позиционная может и не рассматриваться; счёт на пальцах, зарубки, узелки «на память» …);
- ***двоичная система счисления*** (в дискретной математике, информатике, программировании);
- ***троичная система счисления***;
- ***четверичная система счисления*** применяется в вычислительной технике;
- ***десятичная система счисления***;
- двенадцатеричная система счисления (счет дюжинами);
- шестидесятиричная (единицы измерения времени, измерение углов и, в частности, координат долготы и широты).

Смешанная система счисления является обобщением *b*-ричной системы счисления и также зачастую относится к позиционным системам счисления. Основанием смешанной системы счисления является возрастающая последовательность чисел $\mathbf{\{b_k\}_{k=0}^{\infty}}$, и каждое число *x* представляется как линейная комбинация :

$$\mathbf{x=sum(a_k*b_k;\ k=0,...,n-1)}\ ,$$

где на коэффициенты ak (называемые, как и прежде, цифрами) накладываются некоторые ограничения.

Записью числа *x* в смешанной системе счисления называется перечисление его цифр в порядке уменьшения индекса *k*, начиная с первого

ненулевого значения цифры.

В зависимости от вида b_k как функции от k смешанные системы счисления могут быть ***степенными, показательными*** и т.п.

Показательная смешанная система счисления, когда $b_k = b^k$ для некоторого b, совпадает с b-ричной системой счисления.

Наиболее известным примером смешанной системы счисления является представление времени в виде количества суток, часов, минут и секунд. При этом величина *d дней h часов m минут s секунд* соответствует значению d*24*60*60+h*60*60+m*60+s секунд .

В ***факториальной системе счисления*** основанием является последовательность факториалов $b_k = k!$, и каждое натуральное число x представляется в виде

x=sum(dk*k!; k=1,...,n), где 0 ≤ dk ≤ k.

Система счисления майя имела некоторое отличие от обычной позиционной системы счисления. Майя использовали 20-ричную систему счисления за одним исключением: во втором разряде было не 20, а 18 ступеней, то есть за числом (17)(19) сразу следовало число (1)(0)(0). Это было сделано для облегчения расчётов календарного цикла, поскольку (1)(0)(0) = 360 примерно равно числу дней в солнечном году. Для записи основными знаками были точки (единицы) и отрезки (пятёрки).

В ***непозиционных системах счисления*** величина, которую обозначает цифра, не зависит от положения в числе.

Каноническим примером фактически непозиционной системы счисления является ***римская***, в которой в качестве цифр используются латинские буквы. ***Римские цифры*** **-** цифры, использовавшиеся древними римлянами в своей непозиционной системе счисления. Римские цифры появились около 500 лет до нашей эры у этрусков.

Число	**1**	**5**	**10**	**50**	**100**	**500**	**1000**
Римская цифра	**I**	**V**	**X**	**L**	**C**	**D**	**M**

Натуральные числа записываются при помощи повторения этих цифр. Здесь символ I обозначает 1 независимо от места в числе. Например, II=I+I (число 2).

Число	**Римская цифра**	**Пояснения**
0	**отсутствует**	

4	**IV**	**(5-1)**
8	**VIII**	**(5+3)**
9	**IX**	**(10-1)**
31	**XXXI**	**[(10+10)+(10-1)]**
46	**XLVI**	**[(50-10)+(5+1)]**
99	**XCIX**	**[(100-10)+(10-1)]**
666	**DCLXVI**	**[(500+100)+(50+10)+(5+1)]**
1668	**MDCLXVIII**	**[(1000+500+100+50+10)+(5+3)]**
1989	**MCMLXXXIX**	**[(1000+(1000-100)+(50+10+10+10)+(10-1)]**
2009	**MMIX**	**[(1000+1000)+(10-1)]**
3999	**MMMCMXCIX**	**[(1000+1000+1000)+(1000-100)+(100-10)+(10-1)]**

На самом деле, римская система не является полностью непозиционной, так как если большая цифра стоит перед меньшей, то они складываются, например, VI=5+1=6 (принцип сложения), если же меньшая — перед большей, то меньшая вычитается из большей например: IV=5-1=4 (принцип вычитания). Последнее правило применяется только во избежание четырёхкратного повторения одной и той же цифры.

Для правильной записи больших чисел римскими цифрами необходимо сначала записать число тысяч, затем сотен, затем десятков и, наконец, единиц.

Довольно часто, чтобы выделить числа в тексте, над ними рисовали черту. Иногда черту рисовали и сверху, и снизу. В частности, так принято выделять римские цифры в русском рукописном тексте (в типографском наборе это не используют из-за технической сложности).

Существует «сокращённый способ» для записи больших чисел, таких как 1999. Он не рекомендуется, но иногда используется для упрощения. Отличие состоит в том, что для уменьшения цифры слева от неё может писаться любая цифра:

- 999; тысяча M, вычтем 1 (I), получим 999 (IM) вместо CMXCIX;
  следствие: 1999 — MIM вместо MCMXCIX;
- 95; сто C, вычтем 5 (V), получим 95 (VC) вместо XCV;
- 1950; тысяча M, вычтем 50 (L), получим 950 (LM);
  следствие: 1950 — MLM вместо MCML .

Повсеместно записывать число «четыре» как «IV» стали только в XIX веке, до этого наиболее часто употреблялась запись «IIII». Однако запись «IV» можно встретить уже в документах манускрипта «Forme of Cury», датируемых 1390 годом. На циферблатах часов в большинстве случаев

традиционно используется «IIII» вместо «IV», главным образом, по эстетическим соображениям: такое написание обеспечивает визуальную симметрию с цифрами «VIII» на противоположной стороне, а перевёрнутую «IV» прочесть труднее, чем «IIII».

В русском языке римские цифры используются в следующих случаях:

- номер века или тысячелетия: XIX век, II тысячелетие до нашей эры ;
- порядковый номер монарха: Карл V, Екатерина II;
- номер тома в многотомной книге (иногда — номера частей книги, разделов или глав);
- в некоторых изданиях — номера листов с предисловием к книге, чтобы не исправлять ссылки внутри основного текста при изменении предисловия;
- маркировка циферблатов часов «под старину»;
- иные важные события или пункты списка, например: V постулат Евклида, II Мировая война, XXII съезд КПСС и т. п.

В других языках сфера применения римских цифр может иметь особенности, например, в западных странах римскими цифрами иногда записывается номер года.

Греческая система счисления*,** также известная как ***ионийская, или *новогреческая,* - непозиционная система счисления, в которой в качестве символов для счёта употребляют греческие буквы, а также дополнительные символы, такие как **ϛ (стигма), □ (копа) и □ (сампи).** Эта система пришла на смену аттической, или старогреческой, системе, которая господствовала в Греции в III веке до нашей эры. Необходимость сохранять порядок букв ради сохранения их числовых значений привела к относительно ранней (IV век до нашей эры) стабилизации греческого алфавита.

Двоичная система счисления — это позиционная система счисления с основанием 2. В этой системе счисления натуральные числа записываются с помощью всего лишь двух символов (в роли которых обычно выступают цифры 0 и 1).

Двоичная система используется в цифровых устройствах, поскольку является наиболее простой и соответствует требованиям:

- чем меньше значений существует в системе, тем проще изготовить отдельные элементы, оперирующие этими значениями. В частности, две цифры двоичной системы счисления могут быть легко представлены многими физическими явлениями: есть ток — нет тока, индукция магнитного поля больше пороговой величины или нет и т. д.;

- чем меньше количество состояний у элемента, тем выше помехоустойчивость и тем быстрее он может работать. Например, чтобы закодировать три состояния через величину индукции магнитного поля, потребуется ввести два пороговых значения, что не будет способствовать

помехоустойчивости и надёжности хранения информации;

- двоичная арифметика является довольно простой. Простыми являются таблицы сложения и умножения — основных действий над числами;

- возможно применение аппарата алгебры логики для выполнения побитовых операций над числами.

В цифровой электронике одному двоичному разряду в двоичной системе счисления соответствует один двоичный логический элемент (инвертор с логикой на входе) с двумя состояниями (открыт, закрыт); сложение: 1 + 0 = 1; 1 + 1 = 10; 10 + 10 = 100; таблица умножения двоичных чисел: 0 • 0 = 0; 0 • 1 = 0; 1 • 0 = 0;1 • 1 = 1.

Использование двоичной системы при измерении дюймами : при указании линейных размеров в дюймах по традиции используют двоичные дроби, а не десятичные, например: $5\ ^{3}/_{4}''$, 715/16″, 311/32″ и т.д.

Для преобразования из двоичной системы в десятичную используют следующую таблицу степеней основания 2:

512	**256**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**

Начиная с цифры 1, все цифры умножаются на два. Точка, которая стоит после 1, называется двоичной точкой.

Преобразование двоичных чисел в десятичные.

Допустим, вам дано двоичное число 110001. Для перевода в десятичное просто запишите его справа налево как сумму по разрядам следующим образом:

1*20+0*21+0*22+0*23+1*24+1*25=1*1+0*2+0*4+0*8+1*16+1*32=49.

Можно записать это в виде таблицы следующим образом:

512	**256**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
				1	**1**	**0**	**0**	**0**	**1**
				(+32)	**(+16)**				**(+1)**

Точно так же, начиная с двоичной точки, двигайтесь справа налево. Под каждой двоичной единицей напишите её эквивалент в строчке ниже. Сложите получившиеся десятичные числа. Таким образом, двоичное число 110001 равнозначно десятичному 49.

Преобразование методом Горнера.

Для того, чтобы преобразовывать числа из двоичной в десятичную систему данным методом, надо суммировать цифры слева-направо, умножая

ранее полученный результат на основу системы (в данном случае 2). Например:

Двоичное число 1011011 переводится в десятичную систему так: 0*2+**1**=1 >> 1*2+**0**=2 >> 2*2+**1**=5 >> 5*2+**1**=11 >> 11*2+**0**=22 >>22*2+**1**=45 >> 45*2+**1**=91 (в десятичной системе это число будет записано как 91).

Число 101111 переводится в десятичную систему так:
0*2+**1**=1 >> 1*2+**0**=2 >> 2*2+**1**=5 >> 5*2+**1**=11 >> 11*2+**1**=23 >> 23*2+**1**=47
(в десятичной системе это число будет записано как 47).

Преобразование десятичных чисел в двоичные.

Допустим, нам нужно перевести число 19 в двоичное.

Вы можете воспользоваться следующей процедурой: 19 /2 = 9 с остатком 1; 9 /2 = 4 с остатком 1; 4 /2 = 2 с остатком 0; 2 /2 = 1 с остатком 0; 1 /2 = 0 с остатком 1.

Итак, мы делим каждое частное на 2 и записываем в остаток 1 или 0. Продолжать деление надо, пока в делимом не будет 1. Ставим числа из остатка друг за другом, начиная с конца. В результате получаем число 19 в двоичной записи (начиная с конца): 10011.

Двоичные дроби.

Запишем дробь в десятичной системе счисления: $0,357=3/10+5/100+7/1000=3*10^{-1}+5*10^{-2}+7*10^{-3}$.

Обобщаем до системы счисления с основанием q:
$0,a_1a_2...a_n=a_1*q^{-1}+a_2*q^{-2}+...+a_n*q^{-n}$.

Распространяем на двоичный случай:
$0,1100101=1*2^{-1}+1*2^{-2}+0*2^{-3}+0*2^{-4}+1*2^{-5}+0*2^{-6}+1*2^{-7}=1/2+1/4+1/32+1/128$.

Рассмотрим обратную ситуацию. Пусть F – число от 0 до 1 (включая ноль, исключая 1), записанное в каком-то виде. На выходе нужно получить дробь, записанную в позиционной системе счисления по основанию q. Запишем соотношение: $0,x_1x_2x_3...x_m...= F$. Справа – исходная дробь, слева – результат преобразования в виде потенциально бесконечной строки неизвестных. При умножении обеих частей на q, получим: $x_1,x_2x_3...x_m...=F*q$. Можно вычислить F*q, а так как исходно F меньше единицы, F*q окажется меньше q. Поскольку числа равны, соответственно, равны и их целые и дробные части: $x_1=[F*q]$ - целое; $0,x_2x_3...x_m...=\{F*q\}$ - остаток. В результате мы получили значение x_1, а вычисление оставшейся части дроби свели к исходной задаче.

Пример: представим 1/3 в виде пяти разрядной двоичной дроби.

- Исходная задача: $0,x_1x_2x_3x_4x_5 = 1/3$;
- умножаем на 2: $x_1,x_2x_3x_4x_5 = 2/3 => x_1 = 0$; $0,x_2x_3x_4x_5 = 2/3$;
- умножаем на 2: $x_2,x_3x_4x_5 = 4/3 => x_2 = 1$; $0,x_3x_4x_5 = 1/3$;
- умножаем на 2: $x_3,x_4x_5 = 2/3 => x3 = 0$; 0,x4x5 = 2/3;
- умножаем на 2: x4,x5 = 4/3 => $x_4 = 1$; $0,x_5 = 1/3$;

- умножаем на 2: $x_5 = 2/3 => x_5 = ?$;
- По правилу, x_5 должен быть равен 0;
- Но, так как это последний из рассматриваемых битов (из соображений разрядности или точности – не важно), здесь разумно применить округление. В результате $x_5 = 1$.
- Общий результат 0,01011.

ТЕМА 3. ОСОБЕННОСТИ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ И ПРОВЕДЕНИЯ АРИФМЕТИЧЕСКИХ РАСЧЕТОВ В ИНСТРУМЕНТАЛЬНЫХ СРЕДАХ

Требования: студенты должны иметь понятие о представлении чисел в компьютере, точности арифметических расчетов, виде чисел и результатов вычислений в различных инструментальных средах.

В математике считается, что множество чисел бесконечно. Это означает, что число может быть как сколь угодно большим, так и сколь угодно малым. Оно может быть образовано любым количеством цифр. Так, существуют бесконечные дроби и бесконечные трансцендентальные константы (например, число π или число e). Но в реальном мире все конечно. Даже Вселенная имеет границы. Ограничена и память компьютера. В компьютере числа представляются в двоичной системе счисления (1 бит - для изображения 0 или 1). А так как на каждую цифру числа нужно выделить несколько бит, то ***любая вычислительная машина считает с конечной точностью***. Применяются два основных способа представления чисел - с фиксированной и плавающей запятой. Большинство универсальных ЭВМ работает с числами, представленными с плавающей запятой, а большинство специализированных - с фиксированной запятой. Однако целый ряд машин работает с числами в этих двух форматах.

Точность вычислений определяется тем, сколько памяти выделяется на хранение каждого числа. На современных компьютерах используется 64-битный формат двойной точности (соответственно, формат одинарной точности - 32-битный). Это позволяет напрямую (с фиксированной запятой) закодировать 2^{64}=18446744073709551616 целых чисел. На практике приходится работать с гораздо большим количеством числовых значений. Наиболее распространённый путь представления значения числа из строки с цифрами - в виде целого числа - запятая по умолчанию находится в конце строки.

Число с фиксированной запятой - формат представления вещественного числа в памяти ЭВМ в виде целого числа. В общем математическом представлении строка из цифр может быть сколь угодно

длинной, а положение запятой обозначается путём явной записи символа запятой в нужном месте. В системах с представлением чисел в формате с фиксированной запятой существует определённое условие относительно положения запятой. Например, в строке из 8 цифр условие может предписывать положение запятой в середине записи (между 4-й и 5-й цифрой). Таким образом, строка «00012345» обозначает число 1,2345 (нули слева всегда можно отбросить). Название «фиксированная запятая» произошло из-за простой метафоры: между двумя заранее определёнными разрядами ставится запятая, превращая, например, целое число 1234 в дробное 12,34. Иногда говорят «фиксированная точка»: в английской традиции целая часть от дробной отделяется точкой.

Фиксированная запятая.

Будем считать, что разрядная сетка машины имеет постоянное число разрядов - n. При представлении чисел с фиксированной запятой считают, что запятая всегда находится перед старшим разрядом, а все числа, которые участвуют в вычислениях, считаются по абсолютной величине меньше единицы: $|X| < 1$. Введём две характеристики чисел: диапазон изменения и точность представления. Диапазон изменения характеризуется теми пределами, в которых могут находиться числа, с которыми оперирует машина: $|X|_{min}=2^{-n} \le |X| \le 1 - 2^{-n}=|X|_{max}$. Числа , которые выходят за диапазон изменения, в ЭВМ не могут быть представлены точно. Тогда число $2^{-n} \le |X| \le 1 - 2^{-n}$ воспринимается как нуль, а число $|X| > |X|_{max} = 1 - 2^{-n}$ воспринимается как бесконечно большое. Этим двум случаям соответствуют понятия «машинного нуля» и «машинной бесконечности». При оптимальном округлении абсолютная ошибка: $|\Delta X| \le 0{,}5*2^{-n}$.

Минимальная относительная ошибка:

$|\delta X|_{min}=|\Delta X|/|X|_{max}=[0{,}5*2^{-n}]/[1-2^{-n}]$,

так как $1-2^{-n} \approx 1$ при большом «n», то $|\delta X|_{min}= 2^{-(n+1)}$.

Максимальная относительная ошибка:

$|\delta X|_{max}=|\Delta X|/|X|_{min}=[0{,}5*2^{-n}]/[2^{-n}]=0{,}5$.

Ошибка представления числа зависит от величины самого числа и способа округления: $2^{-(n+1)} < |\delta X| < 0{,}5$. Заметим, что для малых чисел ошибка может достигать большой величины.

В экспоненциальной (показательной) записи используют стандартный способ представления чисел. Число считается записанным в стандартном виде, если оно записано в виде $m*q^p$, где $m<q$, называется мантиссой; p — целое называется показатель степени (порядок); q — целое основание системы счисления, например 10.

Нормализованная запись отличного от нуля действительного числа - это запись вида $a=(+ \text{ или } -)m*p^q$, где q - целое число (положительное,

отрицательное или ноль), а m - правильная p-ичная дробь, у которой первая цифра после запятой не равна нулю: $(1/p) \le m < 1$. При этом m называется ***мантиссой*** числа, q - ***порядком*** числа. Мантисса меньше 1, и первая значащая цифра - не ноль (p - основание системы счисления). При таком представлении запятая будет расположена в мантиссе перед первой значащей цифрой, что при фиксированном количестве разрядов, отведённых под мантиссу, обеспечивает запись максимального количества значащих цифр числа, то есть максимальную точность представления числа в компьютере. Такое, наиболее выгодное для компьютера, представление вещественных чисел и называется нормализованным.

Примеры:

1) $3{,}1415926 = 0{,}31415926 * 10^{1}$;
2) $1000 = 0{,}1 * 10^{4}$;
3) $0{,}123456789 = 0{,}123456789 * 10^{0}$;
4) $0{,}00001078 = 0{,}1078 * 8^{-4}$ (порядок записан в 10-й системе);
5) $1000{,}0001_2 = 0{,}10000001_2 * 2^{4}$;
6) $753{,}15 = 0{,}75315 * 10^{3}$;
7) $-0{,}000034 = -0{,}34 * 10^{-4}$.

Так как число «0» не может быть записано в нормализованной форме в том виде, в каком она была определена, то считаем, что нормализованная запись нуля в 10-й системе будет такой: $0 = 0{,}0 * 10^{0}$.

Мантиссу англичане называют по-разному: significand, mantissa, fraction (точнее, fraction - это дробная часть – не совсем мантисса, но близко). По-русски – только мантисса. Показатель степени по-английски будет exponent. Русских вариантов три: собственно показатель степени, порядок, экспонента.

Запись в форме с плавающей запятой похожа на запись чисел в описанном выше стандартном виде, но мантисса и порядок записываются раздельно. ***Мантисса записывается в формате с фиксированной запятой***, подразумеваемой после первой цифры. Запись в форме с плавающей запятой используется, в основном, в электронном представлении чисел, при котором используется основание системы счисления 2, а не 10. Кроме того, в двоичной записи мантисса обычно денормализована, то есть запятая подразумевается до первой цифры, а не после, и целой части вообще не имеется ввиду - так появляется возможность и значение 0 сохранить естественным образом. Запись числа в форме с плавающей запятой позволяет производить вычисления над широким диапазоном величин, сочетая фиксированное количество разрядов и точность. Например, в десятичной системе представления чисел с плавающей запятой (3 разряда) операцию умножения, которую мы бы записали как $0{,}12 \times 0{,}12 = 0{,}0144$, в нормальной форме будет представлена в виде $(1{,}20 * 10^{-1}) \times (1{,}20 \times 10^{-1}) = (1{,}44 \times 10^{-2})$. В формате с фиксированной запятой мы бы получили вынужденное округление

0,120 * 0,120 = 0,014. В итоге, потерян крайний правый разряд числа - данный формат не позволяет запятой «плавать » по записи числа.

Для решения этой проблемы и используется формат представления чисел с использованием плавающей запятой (точки). Основная его идея заключается в следующем: число разделяется на две части — значащую часть (мантиссу) и степень.

Например, число 123 можно записать как $1{,}23*10^2$, число 123000 можно записать как $1{,}23*10^5$, число 0,123 можно записать как $1{,}23*10^{-1}$. При этом возможное число вариантов для передаваемых 64 битами чисел резко возрастает. Правда, точность представления чисел от этого не повышается - она всегда равна номеру последнего знака мантиссы. Но так как нас всегда интересуют результаты ограниченной точности (чаще всего лишь до третьего знака после запятой), то модель чисел с плавающей запятой вполне приемлема.

В общем случае, число имеет форму $s*m*2^p$, где s-знак числа (1 бит) , m- целое число, определяющее мантиссу (под него отведено 53 бита), p - показатель степени (от -1074 до 971). Проанализируем приведенные цифры. Если значимая часть числа может определяться 53 двоичными разрядами, то количество его десятичных разрядов не может превышать $53*\lg(2)\approx15$. Это означает, что точность вычислений ограничена 15 действительными числами. Если длина мантиссы поступившего на обработку числа превышает 15 знаков, то оно просто округляется: 1. 999999999999999999999999 ≈2.

Плавающая запятая.

В ЭВМ с плавающей запятой число представляется в виде

$$\mathbf{X = \pm m_x * q^{\pm p}},$$

где: m_x - мантисса числа; q - основание системы счисления; p - порядок.

Разрядная сетка машины ($n_{разр}$) принимает следующий вид:

<знак числа> <мантисса> <знак порядка> <порядок>.

Это лишь условное изображение основных слогов в числе. Заметим, что в реальной ЭВМ может быть принят любой другой порядок расположения.

Пусть «a» разрядов отведено под изображение мантиссы, а «k» разрядов под изображение порядка. Тогда для двоичной системы и нормализованного вида числа

$$\left| X \right|_{max} = 0{,}111....1*2^{+111...1} = (1-2^{-a})*2^{+r-1};\ r=2^k .$$

$q = 2$; $0.1<m_x<1$ - нормализованная мантисса.

$$|X|_{min}=0,1*2^{-111...1}=2^{-1}*2^{-r+1}.$$

Тогда диапазон чисел равен: $2^{-к} \le |X| \le (1-2^{-a})*2^{+r-1}$.

Абсолютная ошибка представления числа в ЭВМ с плавающей запятой равна: $|\Delta X| \le 0,5*2^{-a}$. Так как $2^{-1} \le |m_x| \le (1-2^{-a})$, то минимальная относительная ошибка: $|\delta X|_{min} = [0,5* 2^{-a}] / [1-2^{-m}] \approx 2^{-(a+1)}$ при а большом; максимальная относительная ошибка: $|\delta X|_{max} = [0,5* 2^{-a}] / [2^{-1}]=2^{-a}$.

Видно, что относительная ошибка в ЭВМ с плавающей запятой не зависит от порядка числа. При этом точность представления больших и малых чисел изменяется незначительно. Теоретически «плавающая запятая» имеет преимущества перед «фиксированной». Но соответствующее устройство получается намного сложнее. К тому же специфика выполнения операций с плавающей запятой требует большего числа микроопераций, что приводит к снижению быстродействия ЭВМ.

Достоинством представления чисел в формате с фиксированной запятой является простота и наглядность представления чисел, а недостатком - относительно небольшой диапазон представления величин с угрозой переполнения на одном конце диапазона и потерей точности вычислений на другом. Эта проблема и привела к изобретению плавающей запятой. Например: если нужна точность в 3 значащих цифры, 4-байтовая фиксированная запятая даёт диапазон в 6 порядков (разница приблизительно 10^6 между самым большим и самым маленьким числами), 4-байтовое число одинарной точности— в 70 порядков.

Число в формате с плавающей запятой занимает в памяти компьютера 4 байта (если это число так называемой двойной точности, то 8 байтов). При этом отдельно выделяются разряды для хранения знака числа, знака порядка, порядка и самой мантиссы.

Чем больше разрядов отводится под запись мантиссы, тем выше точность представления числа. Чем больше разрядов занимает порядок, тем шире диапазон от наименьшего отличного от нуля числа до наибольшего числа, представимого в машине при заданном формате.

Основная причина неточности при использовании чисел с плавающей запятой в том, что компьютер не может работать с бесконечными дробями, которые мы знаем из математики, — для них понадобилось бы бесконечное количество памяти. Это и понятно, мы тоже округляем числа до какого-то знака, когда имеем дело с десятичными дробями. До сих пор не существует удовлетворительной математической теории, которая позволяла бы это делать. С другой стороны, погрешность при операциях с целыми числами оценивается легко. Поэтому одна возможность избавиться от неточностей - это использование чисел с фиксированной запятой, упомянутых выше. Эта возможность нашла наиболее широкое применении в финансовых

программах, где заранее известно нужное количество знаков после запятой. Для других же приложений ограничения чисел с фиксированной запятой часто оказываются проблематичными — с их помощью нельзя представить ни очень большие числа, ни очень маленькие. Вместо этого можно дальше использовать числа с плавающей запятой, но при этом всегда учитывать, что возможны неточности. Так, при выводе результаты надо непременно округлять, причём часто автоматического округления недостаточно, и округление приходится задавать явно. Также надо быть осторожным с операцией сравнения. Можно округлять числа перед сравнением, либо, что более эффективно, смотреть на разницу чисел. Проблема здесь, опять же, состоит в том, что сложно оценить размер возможных погрешностей — неизвестно, с какой максимальной точностью можно считать, чтобы погрешности не попали в результат.

Для чисел с плавающей запятой определены несколько специальных значений, которые весьма непривычны для программистов, привыкших к целочисленным операциям. Так, если взять самое большое целое число и прибавить к нему единицу, произойдёт переполнение, и число станет отрицательным. Если же прибавить единицу к самому большому числу с плавающей запятой, то не произойдёт ровным счётом ничего; в результате мы получим то же самое число. Это явление объясняется ниже. Переполнения можно добиться, к примеру, умножив это число на два. Но результат будет несколько необычным — □< число >□ Inf (от англ. infinity = бесконечность). Аналогичным образом можно получить отрицательную бесконечность — -Inf.

Бесконечность получается и при делении на ноль, причём и здесь она может быть как положительной, так и отрицательной (никакого исключения, как при работе с целыми числами, не возникает). И с ней действительно можно работать (решать). Так, если разделить любое число на бесконечность, получится ноль. Произведение двух бесконечностей опять даёт бесконечность, как и сумма бесконечностей с одинаковым знаком. А вот сумма бесконечностей с разными знаками не определена, результатом получается NaN, другое специальное значение (от англ. Not a Number = не < число >). То же самое выйдет, если попытаться умножить бесконечность на ноль или поделить ноль на ноль. В некоторых языках программирования NaN является ещё и результатом неудачного преобразования строки в число . С NaN тоже можно решать, но результат любой операции будет опять же NaN. Ну и под конец ещё одно необычное явление: если в JavaScript написать 1/0, то результатом будет Inf, а вот 1/-0 вернёт -Inf. Для чисел с плавающей запятой действительно определены два нуля: положительный и отрицательный! К счастью, в программе это обычно не нужно учитывать. Оба нуля при сравнении равны и на выводе они, в большинстве языков программирования, тоже выглядят одинаково. Знак нуля важен только для

операций деления и умножения. Поэтому во многих языках программирования нельзя даже определить константу со значением "− 0", она автоматически преобразуется в положительный ноль.

Одинарная точность (single precision). Знак – 1 бит, показатель степени – 8 бит (смещение 127), мантисса – 23 бита + 1 скрытый. Всего 32 бита. Точность □в знаках□. В двоичных знаках понятно – раз мантисса 24 бита, значит, и точность 24 двоичных знака.

Разберёмся с десятичными. Поскольку для точности □в знаках□ масштаб не имеет значения, умножим мантиссу на 2^{24}. Показатель степени на точность не влияет, так что его можно оставить как есть. Получившееся целое число мантисса представляет точно. Для его представления в десятичном виде нужно: $\log_{10}2^{24} = 24*\log_{10}2 = 7{,}2$ десятичных знака. Это значит, что погрешность двоичного представления числа не превосходит половины седьмого десятичного знака, даже чуть меньше.

Двойная точность (double precision). Знак – 1 бит, показатель степени – 11 бит (смещение 1023), мантисса – 52 бита + 1 скрытый. Всего 64 бита.

Расширенная точность (double extended precision). Стандарт не оговаривает точные значения размеров – только нижние границы, более того, даже не оговаривается, имеет ли мантисса скрытый бит; всё это остаётся на "совести" реализации. На практике встречается следующий вариант (реализован во всех X86-совместимых процессорах): Знак – 1 бит, показатель степени – 15 бит (по стандарту – не меньше 15), мантисса – 64 бита (по стандарту – не меньше 64), скрытого бита нет. Всего 80 бит.

В языках C/C++ числам одинарной точности обычно соответствует тип данных float, двойной точности соответствует тип double, расширенной точности - long double.

Представление в памяти. Естественно, в памяти компьютера двоичное число с плавающей запятой представлено набором битов. Существует много стандартов на форматы представления, но почти все они построены по одному шаблону:

- Знак – плюс (0) или минус (1) – 1 бит
- Показатель степени – целое число – сколько-то бит
- Мантисса – дробь – оставшиеся биты.

Знак числа вынесен в отдельное поле, а мантисса считается всегда положительной.

Нормализованная мантисса. Условием нормализации является $1 <= m < q$. Для двоичной системы это означает $1 <= m < 2$. То есть, если число нормализовано, целая часть M всегда будет равна 1 (ноль не подходит, а других вариантов нет). Значит, можно сэкономить на этом бите – не хранить, а только подразумевать его наличие.

Показатель степени. Показатель степени – целое число, он может быть как положительным, так и отрицательным. Видимо, для того, чтобы не

возиться с отрицательными числами, обычно он хранится в виде беззнакового целого со смещением. А для того чтобы получить реальное значение показателя степени, нужно вычесть из записанного значения некоторую константу:

$1{,}72 * 10^{2} = 1{,}72 * 10^{52-50} = 1{,}72 * 10^{502-500}$

$1{,}72 * 10^{-2} = 1{,}72 * 10^{48-50} = 1{,}72 * 10^{498-500}$

Понятно, что это равносильно введению соответствующего масштабного коэффициента, несложно модифицировать алгоритмы операций над числами так, чтобы они его учитывали.

Можно заметить:

• Если значение показателя степени – 00..00 или 11..11, число считается особым.

• Есть два нуля – положительный и отрицательный, оба – особые числа. Операция сравнения должна быть реализована так, чтобы разные нули были равны друг другу.

• Есть ненормализованные числа.

• Показатель степени хранится как беззнаковое целое со смещением.

• Так же есть две бесконечности и большое количество NaN-ов

• Для сортировки таких чисел не нужно понимать их формат. Достаточно считать старший бит знаком, а всё остальное – беззнаковым целым в двоичной системе счисления – порядок сохранится. Действительно, при сравнении чисел по модулю число с большим показателем степени окажется больше, а если показатели степени равны, большим окажется то, у которого большая мантисса.

Представление чисел и результатов вычислений в различных инструментальных средах.

При проведении расчетов численно Mathcad использует аппаратно реализованные арифметические инструменты самого компьютера, оперируя 64-битными числами с плавающей точкой. Именно отсюда проистекают все особенности и недостатки численных расчетов, которые, на первый взгляд, могут показаться следствием несовершенства самой программы. Если целое число образовано более чем 15 цифрами, то также производится округление до 15 значащих разрядов. Мантисса при этом будет записана в виде десятичной дроби, и у числа появится порядок: $233466785544689652354677564 7754=2.3346678554469*10^{30}$.

Аналогичным образом преобразуются и дробные числа: $0.000000005677345678345566 66=5.6773456783456*10^{-9}$.

То, что на хранение показателя степени числа с плавающей точкой выделяется ограниченное число бит, означает, что невозможно работать с очень большими или очень малыми значениями. ***Максимальное число, которое может участвовать в вычислениях, называется машинной***

бесконечностью. Определить его очень просто. Исходя из описанных принципов представления чисел с плавающей точкой, очевидно, что оно должно иметь максимально возможную мантиссу, умноженную на максимально возможную степень. В двоичном представлении это будет число, действительная часть которого образована 53 единицами, а порядок равен 971^2.

Полученная величина и является приблизительным значением искомой машинной бесконечности. Если при расчете системой будет получено значение , большее машинной бесконечности, то она не сможет разместить его в отведенных 64 битах. Соответственно, при этом возникнет сбой, и будет возвращено сообщение об ошибке : □При попытке вычислить это выражение найдено число со степенью, большей, чем 10^{307}□.

Для представления чисел в Excel используется 15 цифр (знаков). При вводе целого числа, которое содержит более 15 цифр, Excel сохранит его с точностью до 15 значащих цифр, заменив остальные разряды нулями. При вводе чисел, содержащих более 15 знаков после десятичной точки, Excel заменит □лишние□ разряды нулями. При работе в среде Excel, самое большое число, которое можно ввести с клавиатуры, равно $9.9999999999999*10^{307}$. При вводе большего числа Excel воспринимает его как текст и выравнивает по левой стороне ячейки. При выполнении промежуточных вычислений Excel сохраняет числа лежащие в диапазоне (+/-) $1.798*10^{308}$. В том случае, когда промежуточный результат выходит за эти пределы, на экран выводится сообщение об ошибке #ЧИСЛО!.

Ограничение имеется не только на положительное значение показателя степени числа с плавающей точкой, но и на отрицательное. Это означает, что существует число, все числа меньше которого воспринимаются как 0. Это число называется □машинным нулем□.

В Mathcad оно равно $1*2^{-1074}=4,94065645841247*10^{-324}$. В Excel воспринимются, как □машинный ноль□ числа, абсолютное значение которых меньше $2.225*10^{-308}$.

Excel обеспечивает □внутреннюю□ (то есть не отображаемую на дисплее) точность хранения чисел до 15 десятичных знаков, однако при выводе на экран их □изображения□ округляются в соответствии с форматом тех ячеек, в которые они выводятся (округляется только □образ□ числа, выведенного на экран, а само число остается неизменным). В Mathcad по умолчанию до нуля округляются числа, по модулю меньшие 10^{-15}.

Отчего так происходит, если лимит машинного нуля дает возможность оперировать куда меньшими значениями? Все дело в погрешности расчетов, предельная точность которых, ввиду ограничений на длину мантиссы числа, составляет 15 значимых цифр. Округление до 10^{-15} ее сглаживает, в результате чего неудобные и некрасивые ответы появляются реже. Например,

если снизить порог нуля до 10^{-20}, мы при вычислении синуса от π получим следующий результат: $\sin(\pi)= 1.22460635382238*10^{-16}$. В рамках погрешности
численных расчетов ответ был получен верно. Однако для человека, далекого от мира компьютеров, но знающего, что $\sin(\pi)= 0$, он неправилен. Чтобы отсечь погрешность, порог нуля должен составлять 10^{-15} , тогда $\sin(\pi)= 0$. Конечно, 15 знаков мантиссы — это более чем достаточно для большинства расчетов. Но в некоторых случаях ошибки округления способны накапливаться, искажая результат весьма значительно.

Используя в численных расчетах математические функции, важно иметь представление о том, как они реализуются на компьютере. Дело в том, что ввиду особенностей, лежащих в основе задания математических функций, возвращаемые ими значения далеко не всегда являются истинными. Если этого не учитывать, то можно допустить ошибку, которую будет очень и очень сложно.

Основные арифметические операции над числами с плавающей точкой, такие как сложение, умножение или деление, поддерживаются компьютером на аппаратном уровне. То есть их проделывает процессор напрямую. Поэтому скорость выполнения подобных операций может быть крайне высокой. Такие же операции, как вычисление синуса или логарифма, аппаратно не поддерживаются. Причина этого не в том, что реализовать необходимый для этого алгоритм □в железе□ невозможно. Нет, это вполне осуществимая задача (более того, основная аксиома микроэлектроники гласит, что аппаратно можно реализовать любой алгоритм). Просто вводить в процессор соответствующий элемент не имеет смысла, так как создать математические функции очень просто программно, используя только аппаратно поддерживаемые арифметические операции.

В основе реализации таких не выразимых в общем случае через арифметические операции (они называются трансцендентными) математических функций, как синус, косинус или логарифм, лежит одна чрезвычайно важная идея, доказываемая в математическом анализе. Ее суть сводится к тому, что функция, при соблюдении некоторых ограничений, может быть заменена в окрестности данной точки на степенной ряд.

Степенной ряд представляет собой сумму вида

$$a_0*x^0+a_1*x^1+ a_2*x^2+ a_3*x^3+ \ldots+a_n*x^n,$$

где $a_0,a_1,\ldots, a_n$ – некоторые коэффициенты.

Нахождение для функции приближающего ее ряда называется разложением в степенной ряд. Чем больше будет членов разложения, тем точнее будет приближена функция. Также точность приближения зависит от

того, насколько далеко располагается данная точка от той, для которой разложение было проведено. Зная, в какой ряд разлагается функция, написать на основании него алгоритм для определения ее значений не составляет никакого труда.

Итак, математические функции вычисляются при помощи приближенного ряда. Точность таких вычислений равна в идеале точности представления чисел. Это означает, что в 14-15-м знаке мантиссы ошибка будет почти наверняка. Конечно, это неважно в подавляющем большинстве случаев. Но иногда и эта ошибка может стать фатальной. Прежде всего это касается точек разрывов и нулей функций. Например, для любого школьника очевидно, что sin(n*π) при целых n равняется нулю. В Mathcad же при проведении расчета численно равенство sin(n*π)=0 почти наверняка не будет соблюдаться. То же самое можно сказать о значении любой функции практически в любой точке: точным до 15-го знака мантиссы оно не будет.

Формат вывода числовых результатов в Excel.

Окно диалога □Формат ячеек□ (Ctrl+1) позволяет управлять отображением числовых значений. Для вывода результатов вычислений в Excel обычно используют □числовой формат□. Перед открытием окна диалога выделяйте ячейку, содержащую число, которое надо форматировать. В этом случае всегда будет виден результат в поле □Образец□. Не следует забывать о различии между хранимыми и отображаемыми значениями. На хранимые числовые или текстовые значения в ячейках форматы на действуют.

Общий формат. Любое введенное текстовое или числовое значение по умолчанию отображается в формате □Общий□. При этом оно отображается точно так, как было введено в ячейку за исключением трех случаев:

1) длинные числовые значения отображаются в экспоненциальной форме записи или округляются;
2) формат не отображает незначащие нули (456,00 = 456);
3) десятичная дробь, введенная без числа слева от десятичной запятой, выводится с нулем (,23 = 0,23).

Числовые форматы. Этот формат позволяет выводить числовые значения в виде целых чисел или чисел с фиксированной запятой, а также выделять отрицательные числа с помощью цвета.

Десятичная запятая. Показывает, сколько цифр (0 или #) выводится справа и слева от десятичной запятой. Если формат содержит только метки # слева от этого символа, Excel начинает числа, меньшие 1, с десятичной запятой. Чтобы избежать этого, надо использовать 0 вместо # в качестве первой метки цифры слева от десятичной запятой.

Экспоненциальные форматы отображают числа в экспоненциальной записи. Данный формат очень удобно использовать для отображения и вывода очень малых или очень больших чисел. Выводят число в

экспоненциальной записи и вставляют E или e в отображаемое значение, если формат содержит 0 или # справа от E- E+ e- e+. Число символов 0 или # справа от E или e определяет минимальное количество цифр в показателе степени. При использовании E- или e- знак показателя степени выводится только тогда, когда он отрицательный, а при использовании E+ или e+ знак показателя степени выводится всегда

Дробный формат. Выводит дробную часть в виде простой дроби. Количество меток цифр, которые окружают этот символ, определяет точность выводимого значения.

Формат вывода численных результатов в Mathcad.

Вывод результата в определенном формате определяется с помощью вставки **Формат**. Открывается окно, в котором выделяются: **Формат числа** и **Погрешность**. В окне ***Погрешность*** отражены параметры: ***Комплексный порог*** (по умолчанию 10); ***Нулевой порог*** (по умолчанию 15). В окне ***Формат числа*** предлагаются 5 форматов вывода численного результата:

1) ***основной (общий)*** - формат, выбираемый по умолчанию; позволяет произвольным образом определять количество отображаемых десятичных знаков; порядковый (экспоненциальный) порог; визуализация незначащих нулей;

2) ***десятичный*** - результат отображается только в виде десятичной дроби; десятичная часть числа контролируется параметром □число десятичных знаков□; в том случае, если полученный результат имеет целую часть длиннее 15 знаков (или заданного числа десятичных знаков), все цифры, лежащие в нем после этой позиции, будут заменены нулями, если задана визуализация незначащих нулей;

3) ***научный*** - число отображается только со степенью таким образом, чтобы целая часть мантиссы состояла из одного символа; количество десятичных знаков и отображение незначащих нулей результата определяется пользователем; кроме того, существует возможность представления числа в техническом формате (параметр □показывать показатель степени как E(+/-)000)□);

4) ***инженерный*** — формат, очень близкий к научному; единственным отличием является то, что порядок числа должен быть обязательно кратен трем;

5) ***дробь (дробный)*** — формат , отличающийся от других; результат вычислений представляется в виде простой дроби; его точность можно регулировать при помощи параметра □уровень точности□; данный параметр определяет, с точностью до какого десятичного знака будет приближен результат (область его изменения от 0 до 15); по умолчанию, числа, имеющие целую часть, приближаются неправильными дробями; для того, чтобы целая часть была выделена в дробном формате, следует задействовать опцию:□использовать смешанные числа□.

Функции Excel, предназначенные для выполнения операций с результатом вычислений.

Использование аргументов. При использовании в функции нескольких аргументов они отделяются один от другого точкой с запятой. Аргументы функции могут быть числовыми, в частности действительными числами. Аргументы ряда функций могут принимать только логические значения ИСТИНА или ЛОЖЬ. Логическое выражение возвращает значение ИСТИНА или ЛОЖЬ в ячейку или формулу, содержащую это выражение.

ABS (Число) возвращает значение модуля числа (модуль - значение числа без знака (+) или (-). Синтаксис: ABS (Число)).

ЗНАК (Число), аргумент - любое вещественное число. Определяет знак числа. Возвращает 1, если число положительное, ноль (0), если число равно 0, и -1, если число отрицательное.

Операции округления - изменения отображаемого количества десятичных знаков без изменения значения* : *функции ОКРУГЛ, ЦЕЛОЕ, ОТБР, ОКРУГЛВВЕРХ, ОКРУГЛВНИЗ.

Функция ***ОКРУГЛ (ROUND)*** округляет число, задаваемое ее аргументом, до указанного количества десятичных разрядов и имеет следующий синтаксис: **=ОКРУГЛ(число;количество_цифр).** Аргумент **число** может быть числом, ссылкой на ячейку, в которой содержится число, или формулой, возвращающей числовое значение. Аргумент **количество_цифр**, который может быть любым положительным или отрицательным целым числом, определяет, сколько цифр будет округляться. Задание отрицательного аргумента **количество_цифр** округляет до указанного количества разрядов слева от десятичной запятой, а задание аргумента **количество_цифр** равным 0 округляет до ближайшего целого числа.

Примечания:

1) Если число_разрядов больше 0, то число округляется до указанного количества десятичных разрядов справа от десятичной запятой.
2) Если **число_разрядов** равно 0, то число округляется до ближайшего целого.
3) Если **число_разрядов** меньше 0, то число округляется слева от десятичной запятой.

Функция ЦЕЛОЕ (INT) округляет число вниз до ближайшего целого и имеет следующий синтаксис: **ЦЕЛОЕ(Число)**. Аргумент - **число** - это число, для которого надо найти следующее наименьшее целое число.

Функция ОТБР (TRUNC) отбрасывает все цифры справа от десятичной запятой независимо от знака числа. Необязательный аргумент **количество_цифр** задает позицию, после которой производится усечение. Функция имеет следующий синтаксис: **=ОТБР(число;количество_цифр).** Возвращает **число** с количеством знаков после запятой не более чем

число_разрядов . Значение по умолчанию аргумента **число_разрядов 0** (нуль) — отбрасывается дробная часть числа.

Функция ***ОТБР и ЦЕЛОЕ*** подобны в том, что возвращают целые. ***ОТБР*** отбрасывает дробную часть числа. ***ЦЕЛОЕ*** округляет число до ближайшего целого с недостатком с учетом значения дробной части. Эти функции различаются только при использовании отрицательных чисел: ОТБР(-4,3) возвращает -4, но ЦЕЛОЕ(-4,3) возвращает -5, поскольку -5 это ближайшее снизу число.

Функции ***ОКРУГЛ, ЦЕЛОЕ и ОТБР*** удаляют ненужные десятичные знаки, но работают они различно. Функция ***ОКРУГЛ*** округляет вверх или вниз до заданного числа десятичных знаков. Функция ***ЦЕЛОЕ*** округляет вниз до ближайшего целого числа, а функция ***ОТБР*** отбрасывает десятичные разряды без округления. Основное различие между функциями ***ЦЕЛОЕ*** и ***ОТБР*** проявляется в обращении с отрицательными значениями. Если вы используете значение -10,900009 в функции ***ЦЕЛОЕ,*** результат оказывается равен -11, но при использовании этого же значения в функции ***ОТБР*** результат будет равен -10.

Функции ***ОКРУГЛВНИЗ (ROUNDDOWN)*** и ***ОКРУГЛВВЕРХ (ROUNDUP)*** имеют такой же синтаксис, как и функция ***ОКРУГЛ.*** Они округляют значения вниз (с недостатком) или вверх (с избытком).

ОКРУГЛВНИЗ(число;число_разрядов).

Число - любое вещественное число, которое нужно округлить с недостатком.

Число_разрядов - количество цифр, до которого округляется число.

Примечания:

1) Функция ОКРУГЛВНИЗ подобна функции ОКРУГЛ, за тем исключением, что число всегда округляется с недостатком.
2) Если **число_разрядов** больше 0 (нуля), то число округляется с недостатком до заданного количество десятичных разрядов после запятой.
3) Если **число_разрядов** равно 0, то число округляется вниз до ближайшего целого.
4) Если **число_разрядов** меньше 0, то число округляется с недостатком до заданного количеств десятичных разрядов слева от запятой.

ОКРУГЛВВЕРХ(число;число_разрядов).

Число - любое вещественное число, которое нужно округлить с избытком.

Число_разрядов - количество цифр, до которого округляется число.

Примечания:

1) Функция ОКРУГЛВВЕРХ подобна функции ОКРУГЛ, за тем исключением, что округление всегда производится с избытком.
2) Если **число_разрядов** больше 0 (нуля), то число округляется с избытком до заданного количества десятичных разрядов после

десятичной запятой.

3) Если **число_разрядов** равно 0, то число округляется до ближайшего целого.
4) Если **число_разрядов** меньше 0, то число округляется с избытком, с учетом десятичных разрядов слева от десятичной запятой.

Функции *ОКРВНИЗ, ОКРВВЕРХ.* Функции ***ОКРВНИЗ (FLOOR)*** и ***ОКРВВЕРХ (CEILING)*** тоже можно использовать для выполнения операций округления. Функция ***ОКРВНИЗ*** округляет число вниз до ближайшего кратного для заданного множителя, а функция ***ОКРВВЕРХ*** округляет число вверх до ближайшего кратного для заданного множителя.

Эти функции имеют следующий синтаксис: **=ОКРВНИЗ(число;множитель); =ОКРВВЕРХ(число;множитель).**

ОКРВНИЗ(**число**; **точность** или **множитель)**.

Число - это округляемое числовое значение.

Множитель или точность – это кратное, до которого требуется округлить.

Примечания:

1) Если любой из аргументов не число, то ОКРВНИЗ возвращает значение ошибки #ЗНАЧ!.
2) Если число и точность имеют разные знаки, то ***ОКРВНИЗ*** возвращает значение ошибки #ЧИСЛО!.
3) Независимо от знака числа, округление всегда производится с недостатком. Если число уже кратно точности, то никакого округления не производится.

ОКРВВЕРХ(**число**; **точность** или **множитель**).

Число - это округляемое значение.

Точность - это кратное, до которого требуется округлить.

Примечания:

1) Если один из аргументов не является числом, то ОКРВВЕРХ возвращает значение ошибки #ЗНАЧ!.
2) Независимо от знака числа, округление производится с избытком. Если число уже кратно точности, то округления не производится.
3) Если число и точность имеют разные знаки, то функция ***ОКРВВЕРХ*** возвращает значение ошибки #ЧИСЛО!.

ОКРУГЛТ(**число**;**точность** или **множитель**).

Число - округляемое значение.

Точность - точность, с которой требуется округлить число.

Если данная функция недоступна или возвращает ошибку #ИМЯ?, установите и загрузите надстройку **Пакет анализа**:

- В меню **Сервис** выберите команду **Надстройки**.
- В **списке надстроек** выберите **Пакет анализа** и нажмите кнопку **ОК**.
- Выполните инструкции программы установки, если это необходимо.

ОКРУГЛТ производит округление с избытком. Округление производится, если остаток от деления числа на точность больше или равен половине точности.

Значения **число** и **множитель** должны быть числовыми и иметь один и тот же знак.

Для выполнения операций округления можно использовать функции **ЧЁТН (EVEN) и НЕЧЁТ (ODD).** Функция **ЧЁТН** округляет число вверх до ближайшего четного целого числа. Функция **НЕЧЁТ** округляет число вверх до ближайшего нечетного целого числа. Отрицательные числа округляются не вверх, а вниз. Функции имеют следующий синтаксис: **=ЧЁТН(число); =НЕЧЁТ(число).**

ЧЁТН(число). Возвращает число, округленное до ближайшего четного целого. Число — это округляемое значение.

Примечания:

1) Если аргумент число числом не является, то ЧЁТН возвращает значение ошибки #ЗНАЧ!.
2) Независимо от знака числа округление производится с избытком. Если число уже является четным целым, то никакого округления не производится.

НЕЧЁТ(число). Число— округляемое значение. Возвращает число, округленное до ближайшего нечетного целого.

Примечания:

1) Если аргумент число не является числом, то функция **НЕЧЁТ** возвращает значение ошибки #ЗНАЧ!.
2) Независимо от знака числа, округление всегда производится с избытком. Если число является нечетным целым, то округления не происходит.

Для проверки четности или нечетности числа используют функции **ЕЧЁТ, ЕНЕЧЁТ**. Функции имеют следующий синтаксис: **=ЕЧЁТ(число); ЕНЕЧЁТ (число)**.

Функция **ЕЧЁТ(число)** возвращает значение ИСТИНА, если число четно, и значение ЛОЖЬ, если число нечетно.

Функция **ЕНЕЧЁТ (число)** возвращает значение ИСТИНА, если число нечетное, и значение ЛОЖЬ, если число четное.

Если эти функции недоступны или возвращают ошибку #ИМЯ?, установите и загрузите надстройку **Пакет анализа**.

Математические функции Excel.

Функция СУММ (SUM) суммирует множество чисел. Эта функция имеет следующий синтаксис:**=СУММ(числа).** Аргумент числа может включать до 30 элементов, каждый из которых может быть числом, формулой, диапазоном или ссылкой на ячейку, содержащую или возвращающую числовое значение. Функция **СУММ** игнорирует аргументы,

которые ссылаются на пустые ячейки, текстовые или логические значения. Аргументы не обязательно должны образовывать непрерывные диапазоны ячеек.

Функция ПРОИЗВЕД (PRODUCT) перемножает все числа, задаваемые ее аргументами, и имеет следующий синтаксис: **=ПРОИЗВЕД(число1;число2...).** Эта функция может иметь до 30 аргументов. Excel игнорирует любые пустые ячейки, текстовые и логические значения.

Функция ОСТАТ (MOD) возвращает остаток от деления и имеет следующий синтаксис: **=ОСТАТ(число;делитель).** Значение функции **ОСТАТ** - это остаток, получаемый при делении аргумента число на делитель. Если число точно делится на делитель, функция возвращает 0. Если делитель равен 0, функция **ОСТАТ** возвращает ошибочное значение.

Функция КОРЕНЬ (SQRT) возвращает положительный квадратный корень из числа и имеет следующий синтаксис: **=КОРЕНЬ(число).** Аргумент **число** должен быть положительным числом. Если число отрицательное, **КОРЕНЬ** возвращает ошибочное значение.

Функция ЕЧИСЛО (ISNUMBER) определяет, является ли значение числом, и имеет следующий синтаксис: **=ЕЧИСЛО(значение).** Пусть вы хотите узнать, является ли значение в ячейке A1 числом. Следующая формула возвращает значение ИСТИНА, если ячейка A1 содержит число или формулу, возвращающую число; в противном случае она возвращает ЛОЖЬ: =ЕЧИСЛО(A1).

Функция LOG возвращает логарифм положительного числа по заданному основанию. Синтаксис: **=LOG(число;основание).** Если аргумент основание не указан, то Excel примет его равным 10.

Функция LN возвращает натуральный логарифм положительного числа, указанного в качестве аргумента. Эта функция имеет следующий синтаксис: **=LN(число).**

Функция EXP вычисляет значение константы e, возведенной в заданную степень. Эта функция имеет следующий синтаксис: **=EXP(число).** Функция **EXP** является обратной по отношению к **LN**.

Функция ПИ (PI) возвращает значение константы пи с точностью до 14 десятичных знаков. Синтаксис: **=ПИ(число).**

Тригонометрические функции используют углы, выраженные в радианах, а не в градусах.

Функция SIN возвращает синус угла и имеет следующий синтаксис: **=SIN(число).** Здесь число - угол в радианах.

Функция COS возвращает косинус угла и имеет следующий синтаксис: **=COS(число).** Здесь число - угол в радианах.

Функция TAN возвращает тангенс угла и имеет следующий синтаксис: **=TAN(число).** Здесь число - угол в радианах.

Измерение углов в радианах основывается на константе пи и при этом 180 градусов равны пи радиан. Excel предоставляет две функции: **РАДИАНЫ (RADIANS) и ГРАДУСЫ (DEGREES),** чтобы облегчить работу с тригонометрическими функциями.

Для преобразования градусов в радианы используется **функция РАДИАНЫ**, которая имеет следующий синтаксис: **=РАДИАНЫ(угол).** Здесь **угол** - это число, представляющее собой угол, измеренный в градусах. Формула: =РАДИАНЫ(180) возвращает значение 3,14159.

Для преобразования радиан в градусы используется **функция ГРАДУСЫ**, которая имеет следующий синтаксис: **=ГРАДУСЫ(угол).** Здесь **угол** - это число, представляющее собой угол, измеренный в радианах. Формула: =ГРАДУСЫ(3,14159) возвращает значение 180.

Здесь рассмотрены наиболее часто используемые математические функции Excel. Дополнительную информацию о функциях можно найти в окне диалога мастера функций, а также в *справочной системе Excel*. Кроме того, множество математических функций включено в надстройку **Пакет анализа**.

Функции Mathcad, предназначенные для выполнения операций с результатом вычислений.

Ceil(z, y) - возвращает наименьшее кратное y, большее или равное z,

ceil(z) - возвращает наименьшее целое, большее или равное z.

Floor(z, y) - возвращает наибольшее кратное y, меньшее или равное z,

floor(z) - возвращает наибольшее целое, меньшее или равное z.

Round(z, y) - округляет z до ближайшего кратного y,

round(z, n) - округляет z до n разрядов; если n опущено, z округляется до ближайшего целого; если n < 0, z округляется влево от десятичного разделителя.

Trunc(z, y) - возвращает значение trunc(z/y)*y, обычно используемое для правильного масштабирования единицы измерения,

trunc(z) - возвращает целую часть z, удаляя дробную часть.

ТЕМА 4. КОДЫ

Требования: студенты должны иметь понятие о виде представления информации в компьютере в виде кода, знать определения специальных кодов, уметь оперировать с ими.

Любая информация (числа, команды, записи и т. п.) представляется в ЭВМ в виде двоичных кодов фиксированной или переменной длины. Отдельные элементы двоичного кода, имеющие значение 0 или 1, называют разрядами или битами. Двоичный код, состоящий из 8 разрядов носит название байта. Для записи чисел также используют 32-разрядный формат

(машинное слово), 16-разрядный формат (полуслово) и 64-разрядный формат (двойное слово).

Представление целых чисел в компьютере. Целые числа являются простейшими числовыми данными, с которыми оперирует ЭВМ. Для целых чисел существуют два представления: беззнаковое (только для неотрицательных целых чисел) и со знаком. Отрицательные числа можно представлять только в знаковом виде. Целые числа в компьютере хранятся в формате с фиксированной запятой.

Представление целых чисел в беззнаковых целых типах. Для беззнакового представления все разряды ячейки отводятся под представление самого числа. Например, в байте (8 бит) можно представить беззнаковые числа от 0 до 255. Поэтому, если известно, что числовая величина является неотрицательной, то выгоднее рассматривать её как беззнаковую.

Представление целых чисел в знаковых целых типах. Для представления со знаком самый старший (левый) бит отводится под знак числа, остальные разряды - под само число. Если число положительное, то в знаковый разряд помещается 0, если отрицательное -1. Например, в байте можно представить знаковые числа от -128 до 127.

В ЭВМ в целях упрощения выполнения арифметических операций применяют специальные коды для представления чисел. Использование кодов позволяет свести операцию вычитания чисел к арифметическому сложению кодов этих чисел. Применяются ***прямой, обратный и дополнительный коды чисел***.

Выполнение арифметических операций над числами, представленными с фиксированной запятой. Основной особенностью различных методов выполнения арифметических операций является то, что любая операция (сложение, вычитание, умножение, деление и др.) сводится к некоторой последовательности микроопераций, таких как: сложение, сдвиг, передача, преобразование кодов.

Сложение выполняется по правилам сложения чисел в позиционных системах счисления, т. е. эта операция выполняется поразрядно, а возникающий в младших разрядах перенос направляется в старшие разряды. Операции сложения производятся одновременно над всеми разрядами двух слагаемых и продолжаются до тех пор, пока возникают переносы. Возникающие переносы приводят к продолжению операции. Это одна из особенностей позиционных систем. Видим, что собственно операция определения частичной суммы слагаемых выполняется в один приём, а возникающие переносы распространяются на всё более старшие разряды.

Пример: 0,101101 1-е слагаемое

+0,000101	2-е слагаемое
_____	
0,101000	сумма
0,00101	перенос
_____	
0,100010	сумма
0,01	перенос
_____	
0,110010	сумма

Сдвиг. Различают два вида микрооперации сдвига: ***логический сдвиг; арифметический сдвиг***. Логический сдвиг приводит к смещению всех разрядов числа, включая и знак, влево или вправо. При этом освобождающиеся разряды заполняются нулями или единицами. Арифметический сдвиг выполняется над частью числа, часть сдвинутых разрядов теряется. (Очевидно, знаковый разряд должен исключаться из рассмотрения).

Передача. Эта микрооперация предполагает, что некоторый код (число) записывается в соответствующее устройство и вытесняет тот код, который там находился до передачи. Различают два вида передач: запись (с разрушением ранее записанной информации); чтение (без разрушения).

Преобразование. Функция, выполняемая над передаваемыми числами, называется преобразованием. Чаще других в арифметических основах рассматривают инвертирование кода. Это поразрядная микрооперация y_i равно результату сложения по модулю два (+) x_i и $1(1 \leq i \leq n)$, которая выполняется над всеми разрядами одновременно (если $x_i = 0$, то $y_i = 1$; если $x_i = 1$, то $y_i = 0$).

Основное неудобство построения устройств, реализующих арифметические операции, состоит в сложном характере алгоритма вычитания. Для его преодоления в ЭВМ всегда операция выполняется по иным правилам, чем это делается обычно. В его основе лежит операция сложения. Алгоритмы выполнения такого рода операций требуют специальных кодов представления отрицательных чисел.

Прямой код числа.

Представление числа в привычной форме «знак» - «величина», при которой старший разряд ячейки отводится под знак, а остальные - под запись числа в двоичной системе, называется прямым кодом двоичного числа.

Например, прямой код двоичных чисел 1001 и -1001 для 8-разрядной ячейки равен 00001001 и 10001001 соответственно.

Положительные числа в ЭВМ всегда представляются с помощью прямого кода. Прямой код числа полностью совпадает с записью самого

числа в ячейке машины.

Прямой код отрицательного числа отличается от прямого кода соответствующего положительного числа лишь содержимым знакового разряда. В цифровых разрядах пишется модуль положительного или отрицательного числа.

Обозначим изображение числа □X□ в прямом коде
$[X]_{ПК}$. $[X]_{ПК} = [X_+]=X$, если $X\geq 0$; $[X]_{ПК} = 1+[X.]=1-X$, если $X\leq 0$.

Рассмотрим диапазоны представляемых чисел:
$X_{+min} = 0{,}000....0$ - изображение положительного нуля;
$X_{+max} = 0{,}111....1 = 1 - 2^{-n}$; $X_{-min} = 1{,}111....1 = - (1-2^{-n})$; $X_{-max} = 1{,}000....0$ - изображение отрицательного нуля.

Таким образом, нуль имеет двоякое изображение.

Примечания:

1. Перед выполнением операции вычитания чисел с одинаковыми знаками и сложения с разными необходимо сравнить по модулю два кода и, если нужно, сделать перестановку кодов местами, затем можно выполнять собственно операцию вычитания кодов.
2. При выполнении операции умножения отдельно и независимо находятся модули произведений кодов, а знак находится как результат операции сложения по модулю два:

$[X]_{пк} + [Y]_{пк} = \text{sign } Z.$ $\left|Z\right| = \left|X\right| + \left|Y\right|$;
sign Z = sign (X) + sign (Y) или Sz = Sx + Sy .
Собственно умножение выполняется с применением микроопераций сложения и сдвига.

3. Аналогично умножению выполняется операция деления с использованием микроопераций вычитания и сдвига.

Вследствие ряда неудобств в ЭВМ операции вычитания, сложения чисел с разными знаками и деления в прямом коде практически не выполняются.

Обратный и дополнительный коды используются для замены операции вычитания операцией сложения, что упрощает устройство арифметического блока ЭВМ.

К кодам выдвигаются следующие требования:

1) разряды числа в коде жестко связаны с определенной разрядной сеткой;
2) для записи кода знака в разрядной сетке отводится фиксированный, строго определенный разряд.

Например, если за основу представления кода взят один байт, то для представления числа будет отведено 7 разрядов, а для записи кода знака один разряд. Знаковым разрядом обычно является крайний разряд в разрядной сетке. В дальнейшем при записи кода знаковый разряд от цифровых условимся отделять запятой. Если количество разрядов кода не указано,

будем предполагать, что под запись кода выделен один байт.

Обратный код.

Обратным называется код, для которого в знаковом разряде положительного числа пишется □ 0□, в цифровых - модуль числа, а для отрицательного - в знаковом разряде пишется единица, в цифровых - инвертированные разряды исходного числа:

$[X]_{ОК} = [X_+]=X$, если $X \geq 0$;

$[X]_{ОК} = 1+[(1-10^{-n}) - \left| X \right|]=10 -10^{-n} - \left| X \right| =10 -10^{-n}+X$, если $X \leq 0$.

Определим диапазоны чисел:

$X_{+\ min} = 0,00...0$ - положительный нуль;

$X_{+\ max} = 0,111...1 = 1 – 2^{-n}$;

$X_{-\ min} = 1,11...1 = 2 – 2^{-n}+1$;

$X_{-\ max} = 1,00...00 = 1$.

В обратном коде есть два изображения нуля:

□положительный□ нуль: $[X]_{ок} = 0,0...0$;

□отрицательный□ нуль: $[X]_{ок} = 1,11...11$.

При этом:

$X - X =[X_+]_{ок} + [X_-]_{ок} = \left| X_+ \right| +10 - (10)^{-n} - \left| X_- \right| = +10 - (10)^{-n} = 0.$

Единица переноса в знаковом разряде эквивалентна единице младшего разряда. Поэтому при выполнении операции сложения-вычитания необходимо возникающий перенос циклически прибавлять в младший разряд частичного результата.

Дополнительный код.

Дополнительным называется код, в котором для положительного числа в знаковом разряде пишется □0□, в цифровых - модуль числа, а для отрицательного в знаковом разряде пишется □1□, в цифровых - дополнение числа до единицы.

$[X]_{ДК} = [X_+]=X$, если $X \geq 0$; $[X]_{ДК} = 10+X_-$, если $X \leq 0$.

Если некоторое $X_- = -0,x_1x_2...x_n$ нужно представить в дополнительном коде, то $[X]_{ДК} =1. Z_1 Z_2 ...Z_n$, где: $1- 0,x_1x_2...x_n =0, Z_1Z_2...Z_n$.

↑ - знак (отрицательное число).

Диапазоны представлямых чисел:

- $X_{+\ min} = 0,0...0$ - положительный нуль;
- $X_{+\ max} = 0,11...1 = 1-2^{-n}$ — максимальное положительное число;
- $X_{-\ min} = 1,11...1 = 2-2^{-n}$ - минимальное отрицательное число;
- $X_{-\ max} = 1,0...0$ - наибольшее(по модулю) отрицательное число.

Таким образом, нуль имеет единственное представление.

В самом деле, так как $X-X = [X_+]_{дк} + [X_-]_{дк} = 0$, то в дополнительном коде: $\left| X_+ \right| + 10 - \left| X_- \right| = 10$, если в разрядной сетке ЭВМ нет второго

знакового разряда, то это переполнение теряется, и в знаковом разряде будет только нуль.

Для записи дополнительного кода отрицательного числа необходимо в знаковом разряде поставить единицу, проинвертировать все цифровые разряды числа и прибавить единицу в младший разряд. Это также правило перевода из дополнительного кода в прямой код.

Дополнительный код (англ. two's complement, иногда twos-complement) - наиболее распространенный способ представления отрицательных целых чисел в компьютерах. Он позволяет заменить операцию вычитания на операцию сложения, чем упрощает архитектуру ЭВМ.

Дополнительный код положительного числа равен прямому коду этого числа. Дополнительный код отрицательного числа m равен (2^k - m), где k - количество разрядов в ячейке. Для отрицательного числа дополнительный код образуется путем получения обратного кода и добавления к младшему разряду единицы.

Как уже было отмечено, при представлении неотрицательных чисел в беззнаковом формате все разряды ячейки отводятся под само число. Например, запись числа 243 в одном байте при беззнаковом представлении будет выглядеть следующим образом: «1 1 1 1 0 0 1 1». При представлении целых чисел со знаком старший (левый) разряд отводится под знак числа, и под собственно число остаётся на один разряд меньше. Поэтому, если приведённое выше состояние ячейки рассматривать как запись целого числа со знаком, то для компьютера в этой ячейке записано число -13 (243+13=256=2^8).

При записи числа в прямом коде старший разряд является знаковым разрядом. Если его значение равно 0 - то число положительное, если 1 - то отрицательное. В остальных разрядах записывается двоичное представление модуля числа. ***Дополнительный код используется для упрощения выполнения арифметических операций.*** Если бы вычислительная машина работала с прямыми кодами положительных и отрицательных чисел, то при выполнении арифметических операций следовало бы выполнять ряд дополнительных действий. Например, при сложении нужно было бы проверять знаки обоих операндов и определять знак результата. Если знаки одинаковые, то вычисляется сумма операндов и ей присваивается тот же знак. Если знаки разные, то из большего по абсолютной величине числа вычитается меньшее и результату присваивается знак большего числа. ***При таком представлении чисел (в виде только прямого кода) операция сложения реализуется через достаточно сложный алгоритм. Если же отрицательные числа представлять в виде дополнительного кода, то операция сложения, в том числе и разного знака, сводится к их***

поразрядному сложению.

Для компьютерного представления целых чисел обычно используется один, два или четыре байта, то есть ячейка памяти будет состоять из восьми, шестнадцати или тридцати двух разрядов соответственно.

Алгоритм получения дополнительного кода отрицательного числа.

Для получения дополнительного k-разрядного кода отрицательного числа необходимо:

- модуль отрицательного числа представить прямым кодом в k двоичных разрядах;
- значение всех бит инвертировать: все нули заменить на единицы, а единицы на нули (таким образом, получается k-разрядный обратный код исходного числа);
- к полученному обратному коду прибавить единицу.

Пример:

Получим 8-разрядный дополнительный код числа (- 52):

00110100 — число 52;

10110100 — число -52 в прямом коде;

11001011 - число -52 в обратном коде;

11001100 - число -52 в дополнительном коде.

При записи числа в дополнительном коде старший разряд является знаковым. Если его значение равно 0, то в остальных разрядах записано положительное двоичное число, совпадающее с прямым кодом. Если же знаковый разряд равен 1, то в остальных разрядах записано отрицательное двоичное число, преобразованное в дополнительный код . Для получения значения отрицательного числа все разряды, кроме знакового, инвертируются, а к результату добавляется единица. Обратное преобразование, то есть перевод из дополнительного в прямой код, осуществляется аналогично.

Двоичное 8-разрядное число со знаком может представлять любое целое в диапазоне от −128 до +127. Если старший разряд равен нулю, то наибольшее целое число, которое может быть записано в оставшихся 7 разрядах равно $2^7 - 1$, что равно 127.

Особенности сложения чисел в обратном и дополнительном кодах.

При сложении чисел в дополнительном коде возникающая единица переноса в знаковом разряде отбрасывается. При сложении чисел в обратном коде возникающая единица переноса в знаковом разряде прибавляется к младшему разряду суммы кодов. Если результат арифметических действий является кодом отрицательного числа, необходимо преобразовать его в прямой код. При этом обратный код преобразуется в прямой заменой цифр во всех разрядах, кроме знакового на противоположные. Дополнительный код

преобразуется в прямой так же, как и обратный, с последующим прибавлением единицы к младшему разряду.

Двоично-десятичный код - форма записи целых чисел, когда каждый десятичный разряд числа записывается в виде его четырёхбитного двоичного кода. Например, число 311 будет записано в двоичном коде как «100110111», а в двоично-десятичном «0011 0001 0001».

Преимущества:

1) Упрощён вывод чисел на индикацию — вместо последовательного деления на 10 требуется просто вывести на индикацию каждый полубайт. Аналогично, проще ввод данных с цифровой клавиатуры.

2) Для дробных чисел (как с фиксированной, так и с плавающей запятой) при переводе в читаемый десятичный формат и наоборот не теряется точность.

3) Упрощены умножение и деление на 10, а также округление. По этим причинам двоично-десятичный формат применяется в калькуляторах — калькулятор в простейших арифметических операциях должен выводить в точности такой же результат, какой подсчитает человек на бумаге.

Недостатки:

1) Усложнены арифметические операции.
2) Требует больше памяти.
3) В двоично-десятичном коде 8421-BCD существуют запрещённые комбинации битов:

Запрещённые в 8421-BCD битовые комбинации		
1010	1011	1100
1101	1110	1111

Запрещённые комбинации возникают обычно в результате операций сложения, так как в 8421-BCD ипользуются только 10 возможных комбинаций 4-битового поля вместо 16.

Поэтому, при сложении и вычитании чисел формата 8421-BCD действуют следующие правила:

1) При сложении двоично-десятичных чисел каждый раз, когда происходит перенос бита в старший полубайт, необходимо к полубайту, от которого произошёл перенос, добавить корректирующее значение 0110.

2) При сложении двоично-десятичных чисел каждый раз, когда встречается недопустимая для полубайта комбинация, необходимо к каждой недопустимой комбинации добавить корректирующее значение 0110 с разрешением переноса в старшие полубайты.

3) При вычитании двоично-десятичных чисел, для каждого полубайта,

получившего заём из старшего полубайта, необходимо провести коррекцию, отняв значение 0110.

ТЕМА 5. АЛГЕБРА ЛОГИКИ

Требования: студенты должны владеть понятиями элементарного высказывания значений ИСТИНА (TRUE) и ЛОЖЬ (FALSE), знать, что такое инверсия (логическое НЕ, отрицание), конъюнкция (логическое И, логическое умножение), дизъюнкция (логическое ИЛИ, логическое сложение), импликация (следование), пользоваться таблицами их истинности, производить вычисление простых логических выражений, знать порядок логических действий, понимать простейшие законы логики, представлять себе приложение законов логики как в повседневной жизни, так и в информатике.

Общие сведения.

Слово алгебра обычно обозначает определенную область математики. Однако у этого слова есть и другое значение. Этим словом также называют любую формальную систему, обладающую строго определенным набором свойств; ограничимся определением частного случая.

Алгеброй называется множество A элементов, имеющее следующие свойства:

1. a+b=b+a;
2. a+(b+c)=(a+b)+c;
3. a*(b*c)=(a*b)*c.
4. всегда найдется разность x такая, что a+x=b.
5. a*(b+c)=(a*b)+(a*c).
6. (a+b)*c=(a*c)+(b*c).
7. a*0=0*a=0. 8. a*1=1*a=1.

Среди задач, для решения которых используют компьютер, немало таких, которые принято называть логическими. Все знают шуточную задачу о перевозке козла, волка и капусты с одного берега на другой. В этой задаче властвует не арифметика, а умение логически рассуждать. Человек прибегает к логике, когда составляет расписания, распутывает противоречивые показания или составляет инструкции. В логических задачах исходными данными являются не только и не столько числа, а сложные логические суждения, подчас весьма запутанные. Эти суждения и связи между ними бывают иногда столь противоречивы, что для их разрешения привлекают вычислительные машины.

Логика (греч.– слово, рассуждение) – наука о правильном мышлении, которая регламентирует формы и методы интеллектуальной познавательной деятельности, формализуемой с помощью языка. Одна из главных задач логики – определить, как прийти к выводу из предпосылок. Логика служит

базовым инструментом почти любой науки. Основателем логики считают Сократа. Сократ из Афин (469 - 399 до н.э.) – знаменитый античный философ, учитель Платона, воплощенный идеал истинного мудреца в исторической памяти человечества. Учение Сократа было устным; все свободное время он проводил в беседах с приезжими и местными гражданами, политиками и обывателями, друзьями и незнакомыми на различные темы, например, что есть добро и что – зло, что прекрасно, а что безобразно, что добродетель и что порок, как приобретается знание и т.д. Позднее из логики стала выделяться самостоятельная часть - математическая логика, изучающая основания математики и принципы построения математических теорий.

Математическая логика – логика умозаключений, использующая математические методы. У истоков математической логики стоял Готфрид Вильгельм Лейбниц (1646–1716). Математическую логику в алгебру суждений превратил англичанин Джордж Буль (1815–1864). ***Булева алгебра*** (алгебра логики, алгебра суждений) – раздел математики, в котором изучаются логические операции над высказываниями.

Определение 1. ***Булевыми величинами*** (или булевыми константами) ***называются два заранее выбранных разных символа***. По традиции применяются символы 0 и 1. Но надо понимать, что формулы булевой алгебры будут работать независимо от того, как обозначить булевы величины и какой смысл им придать. Например, в электронике это может быть наличие или отсутствие потенциала в +5 вольт в определенной точке схемы, при доказательстве математической теоремы - суждения □истинно□ и □ложно□, а в экспертной системе - ответы □да□ и □нет□.

Определение 2. ***Булевыми переменными*** называются переменные, которые могут принимать значения булевых величин.

Границы применимости. Для того, чтобы некоторую величину можно было обозначать булевой переменной, должны выполняться следующие ограничения:

1. Величина должна принимать два возможных состояния, но не более того.
2. В любой момент времени величина не может принимать оба состояния одновременно.
3. В любой момент времени величина не может принимать ни одного состояния.
4. Если рассматриваются несколько таких величин, то допускается, чтобы каждая из них принимала одно из двух состояний независимо.
5. Не допускается применять одну пару состояний для одной величины, а для другой – другую.

Правила 1, 2, 3 и 5 должны обязательно выполняться все. ***Если не***

выполняется хотя бы одно из них, алгебра логики не применима. Правило 4 поясняет одну ситуацию, когда могут возникнуть сомнения насчет применения алгебры логики.

Булева алгебра весьма распространена. Принцип работы большинства компьютеров основан на ней. Большинство формул математики могут быть только истинными или ложными, так что булева алгебра применима и почти ко всей математике. Оказывается, что и в обыденной речи алгебра логики вполне применима, хотя не везде и не всегда. Таким образом, булева алгебра полезна, но не претендует на сверхуниверсальность. Это - инструмент, который может оказаться удобен для решения определенных задач, для других - неудобен, а для третьих - вообще неприменим.

Понятие высказывания. Операции над простыми высказываниями.

Основным понятием, используемым в алгебре логики, является высказывание. Объектами, с которыми работает алгебра высказываний, являются повествовательные предложения, относительно которых можно сказать, истинны они или ложны. Простым высказыванием называют повествовательное предложение, относительно которого имеет смысл говорить, истинно оно или ложно.

Истинным высказываниям ставится в соответствие цифра □1□ или слово ИСТИНА (TRUE), а ложным – цифра □0□ или слово ЛОЖЬ (FALSE). Условимся обозначать высказывания большими буквами и, следуя Джорджу Булю, истинное (true) высказывание А обозначим так: А=1. В том случае, когда А– ложное (false) высказывание, будем писать: А=0. Запись x=1 означает, что высказывание x истинно. Логическими значениями высказываний является □истина□ и □ложь□. Считается, что каждое высказывание либо истинно, либо ложно и ни одно высказывание не может быть одновременно истинным и ложным. Запись x=0 означает, что высказывание x ложно. Запись X=Y означает, что высказывания X и Y эквивалентны.

Приведем примеры высказываний: 1) Москва — столица России;
2) число 27 является простым; 3) Волга впадает в Каспийское море. Высказывания 1 и 3 являются истинными. Высказывание 2– ложным, потому что число 27 составное 27=3*3*3.

Следующие предложения высказываниями не являются: 1) давай пойдем гулять; 2) 2*x>8; 3) $a*x^2+b*x+c=0$; 4) который час?

Подчеркнем еще раз, что ***отличительным признаком высказывания является свойство быть истинным или ложным***, последние четыре предложения этим свойством не обладают. Невозможно отнести неравенство 2 или уравнение 3 к высказываниям, пока не определено значение x. При x=3 высказывание □2*3>8□ ложно, а при x=5 □2*5>8□– истинно.

Все высказывания подразделяются на простые и сложные. Сложные высказывания образуются из простых высказываний. ***Задача исчисления***

высказываний сводится к определению истинности или ложности сложного высказывания при конкретном наборе истинности или ложности простых высказываний, на которых строится сложное высказывание.

Из простых высказываний можно строить сложные, называемые составными высказывания, соединяя простые логическими операциями. Над простыми высказываниями определены следующие операции:

1) логическое отрицание (NOT);
2) логическое умножение (AND);
3) логическое сложение (OR);
4) логическое следование или импликация;
5) эквивалентность.

Условием называется логическое высказывание, которое может принимать два значения: истина и ложь. В зависимости от его значения определяется дальнейший ход действий. В математике и технике условия формулируются более строго и содержат специальные операции сравнения (>, <, =). Примеры использования условий в математике: если x>0 , то модуль числа равен самому числу; если в линейной функции Y=KX + B коэффициент B = 0, то прямая проходит через начало координат. Анализ условий используется в различных областях техники: если температура воды равна 100^0, то вода переходит в газообразное состояние; если плотность тела больше 1000 кг/м^3, то оно тонет в воде. В электрических схемах цифровых машин простые высказывания соответствуют сигналам, поступающим на вход схемы, а сложные высказывания – выходным сигналам схемы. Такие схемы называются комбинационными. Если сигнал есть, то значение истинности высказывания, которое он изображает, равно 1. Если сигнал отсутствует, то значение истинности высказывания, изображенного сигналом, равно 0.

Переменные, которые могут принимать лишь одно из двух значений 0 или 1 (false или true), называют логическими переменными.

Логические переменные, соответствующие входным сигналам, являются независимыми (входными) переменными, так как значение каждой из них не зависит от значений других логических переменных. ***Логические*** переменные, соответствующие выходным сигналам, являются зависимыми от входных переменных, или, как говорят, ***являются функциями входных переменных.*** Поскольку ***значения этих функций могут быть либо 0, либо 1,*** **то** ***их называют переключательными, или логическими функциями (ЛФ)***.

Истинность составных высказываний, образованных в результате выполнения каких-либо логических операций над простыми высказываниями, зависит только от истинности исходных высказываний.

Описание или представление логической функции может быть задано разными способами: в виде таблицы истинности или в аналитической форме, в виде формулы. Таблица истинности – это таблица, устанавливающая соответствие между всеми возможными наборами логических переменных, входящих в логическую функцию, и значениями функции. В общем случае логическая функция может иметь n входов или аргументов, которые обычно обозначают x_1, x_2, …., x_n.

Запись $y = f(x_1, x_2, \ldots, x_n)$ означает, что логическая переменная y является n-местной логической функцией своих аргументов x_1, x_2, …., x_n. Совокупность значений x_1, x_2, …., x_n представляет n-разрядное двоичное число. Количество разных значений этого числа, то есть наборов входных значений функции, равно 2^n. В таблице истинности каждому набору значений переменных x_1,x_2, …., x_n ставится в соответствие значение функции $y = f(x_1, x_2, \ldots, x_n)$ на этом наборе, либо прочерк, если на этом наборе значение функции не определено. В последнем случае строку с безразличным набором входных переменных можно и не включать в таблицу истинности.

Полностью определенной ЛФ называется логическая функция, значения которой определены для всех наборов своих аргументов. Иначе ЛФ называется неполностью определенной.

Логическая функция двух аргументов определена на четырех наборах (0,0), (0,1), (1,0) и (1,1). Каждому набору аргументов приписывается номер i, соответствующий двоичному числу, определяющему данный набор. Записью $f_i(x_1, x_2)$ обозначают значение функции, если набор ее аргументов определяет двоичное число со значение i.

Среди всех возможных логических функций выделяют ***базисные*** логические функции – это набор логических функций, через которые можно выразить любые другие логические функции. Таких наборов базисных функций всего 4:

1) логическое умножение, или **конъюнкция** - логическая связка □И□. Функция двух аргументов. Обозначается знаками *, или ∩, или &. **Конъюнкция** □А и В□ **истинна,** когда истинны оба суждения А и В. **Конъюнкция** □А и В□ **ложна,** когда ложны суждения А или В. Для любого числа аргументов функция логического умножения (конъюнкции) имеет значение истинно, если одновременно истинны значения всех ее аргументов.

2) логическое сложение, или **дизъюнкция** - логическая связка □ИЛИ□. Функция двух аргументов. Обозначается знаками +, или □. **Дизъюнкция** □А или В□ **истинна,** когда истинно хотя бы одно из суждений А или В. **Дизъюнкция** □А или В□ **ложна,** когда ложны оба суждения А и В. Для любого числа аргументов функция логического сложения (дизъюнкции) имеет значение **истинно**, если **истинно** значение **хотя бы одного** ее

аргумента.

3) отрицание, или **инверсия** - функция НЕ (дополнение). Функция отрицания является функцией одного аргумента. Обозначается А. Отрицание **истинно**, когда ложно суждение А. Отрицание **ложно**, когда **истинно** суждение А.

4) **импликация** - логическое следование А → В. **Импликация** А → В **ложна,** когда посылка А истинна, а следствие В ложно. **Импликация** А → В **истинна,** когда истинно следствие, либо ложны и следствие, и посылка.

Базисные функции используются для представления через них в аналитическом виде, т.е. в виде формулы, любых логических функций. При аналитическом способе представления логическая функция задается в виде формулы, которая содержит обозначения переменных и знаки операций конъюнкции, дизъюнкции и отрицания. Порядок их выполнения следующий: сначала выполняется операция НЕ, затем И и в последнюю очередь ИЛИ. С помощью построения таблиц истинности можно найти значение произвольной логической функции, доказать или опровергнуть утверждение об эквивалентности различных формул.

В алгебре логики доказано, что любую логическую функцию можно выразить только через комбинацию логических операций И, ИЛИ и НЕ. Для приведения логических выражений к эквивалентным, но более простым в записи, используют ряд логических законов. Для преобразования формул (например, с целью замены на эквивалентную, но короче записываемую) используют равенства, имеющие в алгебре логики силу закона. Равносильными (или эквивалентными) называются такие логические высказывания, таблицы истинности для которых совпадают.

Закон тождества. Сформулирован древнегреческим философом Аристотелем. Закон утверждает, что мысль, заключенная в некотором высказывании, остается неизменной на протяжении всего рассуждения, в котором это высказывание фигурирует.

Закон противоречия говорит о том, что никакое предложение не может быть истинно одновременно со своим отрицанием. Например, это яблоко спелое и это яблоко неспелое или $x \cap x = 0$.

Закон исключенного третьего говорит о том, что для каждого высказывания имеются лишь две возможности: это высказывание либо истинно, либо ложно. Третьего не дано. Например, Сегодня я либо получу пять, либо не получу или $x \cup x = 1$.

Закон двойного отрицания (двойной инверсии) заключается в том, что отрицать отрицание какого-нибудь высказывания – то же, что утверждать это высказывание. Например, Неверно, что 2*2<>4 или (x)=x.

Законы идемпотентности (равносильности) говорят о том, что в алгебре логики нет показателей степеней и коэффициентов. Конъюнкция

одинаковых «сомножителей» равносильна одному из них: x∩ x = x. Дизъюнкция одинаковых «слагаемых» равносильна одному из них: x □x = x.

Законы коммутативности и ассоциативности говорят о том, что конъюнкция и дизъюнкция аналогичны одноименным знакам умножения и сложения чисел.

Переместительный (коммутативный) закон*:* x□y = y□x; x∩y = y ∩x. Сочетательный (ассоциативный) закон*:* (x □ y) □ z = x □(y □ z) = x□y□z; (x∩y) ∩ z= x∩(y∩z) = x∩y∩z.

Законы дистрибутивности говорят о том, что логическое сложение и умножение равноправны по отношению к дистрибутивности: не только конъюнкция дистрибутивна относительно дизъюнкции, но и дизъюнкция дистрибутивна относительно конъюнкции.

Распределительный (дистрибутивный) закон: x□(y∩z)=(x□y)∩(x□z); x∩(y □z)=(x∩y)□(x∩z).

Законы де Моргана показывают как отрицаются высказывания (Август де Морган (1806–1871)– шотландский математик и логик).

Эти законы можно выразить в следующих кратких словесных формулировках:

- отрицание логического произведения эквивалентно логической сумме отрицаний множителей: □(x∩y)=(□x)□(□y);
- отрицание логической суммы эквивалентно логическому произведению отрицаний слагаемых : □(x□y)=(□x)∩(□y).
- Законы поглощения показывают как упрощать логические выражения при повторе операнда: x□ (x∩y) = x; x∩(x□y) = x.
- Законы поглощения констант утверждают, что ложь не влияет на значение логического выражения при дизъюнкции: x □0=x;
  а *истина– при конъюнкции*: x∩1 = x.
- Законы исключения констант : x□1= 1; x∩ 0 = 0.
- Закон исключения (склеивания): (x∩y) □(□x∩y)=y; (x□y) ∩(□x□y)=y.
- Правило перевертывания (закон контрапозиции): x → y; y → x.

Представленные выше равенства, устанавливаемые, например, с помощью таблиц истинности, позволяют уже без помощи таблиц получить другие равенства. В этом случае методом получения равенств являются так называемые тождественные преобразования, которые меняют формулу, но не функцию, реализуемую этой формулой (так как каждый шаг такого преобразования означает переход к эквивалентной формуле). Всякая функция алгебры логики может быть выражена формулой, записываемой с помощью символов.

Конъюнкция, дизъюнкция, логическое отрицание в совокупности с законами алгебры логики и возможностями тождественных преобразований логических функций имеет принципиальное значение для обработки

информации центральным процессором ЭВМ. Процессор обрабатывает информацию, закодированную в двоичном алфавите и имеющую форму электрических сигналов, посредством электронных схем. Различные элементы этих схем реализуют перечисленные основные логические функции.

Рассмотрим возможности реализации различных логических операций в инструментальных средах Excel и Mathcad.

Основные логические функции в Mathcad.

Логические операторы Mathcad расположены на панели Boolean(Булевы). Таких операторов 10. Основные логические функции: ***инверсия, конъюнкция, дизъюнкция, сложение по модулю «два».*** На той же панели расположены операторы сравнения: = - равно; # - неравно; > - больше; => - равно или больше; < - меньше; <= - равно или меньше. Результатом выполнения этих операторов будет либо **1** (**ИСТИНА**) или **0** (**ЛОЖЬ**).

Логические функции Excel.

Логические выражения используются для записи условий, в которых сравниваются числа, функции, формулы, текстовые или логические значения. Любое логическое выражение должно содержать по крайней мере один ***оператор сравнения***, который определяет отношение между элементами логического выражения. Результатом логического выражения является логическое значение **ИСТИНА** (**1**) или логическое значение **ЛОЖЬ** (**0**).

***Операторы сравнения*: = Равно; > Больше; < Меньше; >= Больше или равно; <= Меньше или равно; <> Не равно.**

Функция **ЕСЛИ (IF)** задает логическую проверку условий, записанных в качестве аргументов**.** Синтаксис: **=ЕСЛИ(логическое_выражение (тест); значение_если_истина;значение_если_ложь).** В качестве аргументов функции ЕСЛИ можно использовать другие функции.

Вложенные функции ЕСЛИ. Иногда бывает очень трудно решить логическую задачу только с помощью операторов сравнения и функций И, ИЛИ, НЕ. В этих случаях можно использовать вложенные функции ЕСЛИ. Например, в формуле используются три функции ЕСЛИ: **=ЕСЛИ(A1=100; □Всегда□;ЕСЛИ(И(A1>=80;A1<100);□Обычно□;ЕСЛИ(И(A1>=60;A1<80); □Иногда□; □Никогда□)))**. Если значение в ячейке A1 является целым числом, формула читается следующим образом: □Если значение в ячейке A1 равно 100, возвратить строку □Всегда□. В противном случае, если значение в ячейке A1 находится между 80 и 100, возвратить □Обычно□. В противном случае, если значение в ячейке A1 находится между 60 и 80, возвратить строку □Иногда□. И, если ни одно из этих условий не выполняется, возвратить строку □Никогда□. Всего допускается до 7 уровней вложения функций ЕСЛИ.

Функция **ИСТИНА (TRUE)**. Задается логическое значение ИСТИНА(TRUE). Функция ИСТИНА() не требует аргументов и всегда

возвращает логическое значение ИСТИНА(TRUE). НЕ(ИСТИНА())=ЛОЖЬ.

***Функция* ЛОЖЬ (FALSE).** Задается логическое значение ЛОЖЬ (FALSE). Функция ЛОЖЬ() не требует аргументов и всегда возвращает логическое значение ЛОЖЬ (FALSE).

Функции ИСТИНА (TRUE) и ЛОЖЬ (FALSE) предоставляют альтернативный способ записи логических значений ИСТИНА и ЛОЖЬ.

Синтаксис:**=ИСТИНА(); =ЛОЖЬ().** Например, =ИСТИНА(TRUE) возвращает значение ИСТИНА; =НЕ(ИСТИНА()) возвращает значение ЛОЖЬ; =НЕ(ЛОЖЬ()) возвращает значение ИСТИНА.

***Функция* ЕПУСТО.** Если нужно определить, является ли ячейка пустой, можно использовать функцию ЕПУСТО (ISBLANK), которая имеет следующий синтаксис:**=ЕПУСТО(значение).** Аргумент значение может быть ссылкой на ячейку или диапазон. Если значение ссылается на пустую ячейку или диапазон, функция возвращает логическое значение ИСТИНА, в противном случае ЛОЖЬ.

***Основные логические функции*: инверсия(НЕ), конъюнкция (И), дизъюнкция ИЛИ).**

***Инверсия — логическое отрицание «*НЕ*»(«*NOT*»).* Функция НЕ** имеет только один аргумент и возвращает обратное логическое значение: меняет значение своего аргумента на противоположное логическое значение и обычно используется в сочетании с другими функциями. Синтаксис: **=НЕ(логическое_значение).** Логическое значение - любое значение для преобразования. Примеры: =НЕ(А), где А=ИСТИНА (TRUE) преобразуется в неА=ЛОЖЬ (FALSE); А=ЛОЖЬ (FALSE) преобразуется в неА=ИСТИНА (TRUE).

***Конъюнкция - логическое «*И*» («*AND*»*). Возвращает значение ИСТИНА (TRUE), если все аргументы имеют значение ИСТИНА (TRUE). Синтаксис: **И (Логическое значение 1; Логическое значение 2 ... Логическое значение 30)**. Логическое значение 1; Логическое значение 2 ... Логическое значение 30 - проверяемые условия. Если хотя бы один элемент имеет значение ЛОЖЬ (FALSE), возвращается значение ЛОЖЬ (FALSE). Примеры: Если А=ИСТИНА, а В=ЛОЖЬ: =И(А;В) возвращает значение ЛОЖЬ (FALSE). Если А=ЛОЖЬ, а В=ИСТИНА: =И(А;В) возвращает значение ЛОЖЬ (FALSE). Если А=ЛОЖЬ и В=ЛОЖЬ: =И(А;В) возвращает значение ЛОЖЬ (FALSE). Если А=ИСТИНА и В=ИСТИНА: =И(А;В) возвращает значение ИСТИНА (TRUE). Аргументы представляют собой либо логические выражения (ИСТИНА, 1<5, 2+3=7, В8<10), которые возвращают логические значения, либо массивы (А1:С3) логических значений. Все условия могут иметь значение ИСТИНА (TRUE) или ЛОЖЬ (FALSE). Если в качестве параметра указан диапазон, функция использует значение из текущего столбца или строки диапазона. Значение ИСТИНА (TRUE) возвращается, если значения всех ячеек диапазона возвращают результат

ИСТИНА (TRUE).

Дизъюнкция - логическое «ИЛИ» («OR»).Возвращает значение ИСТИНА (TRUE), если хотя бы один из аргументов имеет значение ИСТИНА (TRUE). Возвращает значение ЛОЖЬ (FALSE), если все аргументы имеют значение ЛОЖЬ (FALSE). Синтаксис: **ИЛИ(Логическое значение 1; Логическое значение 2 ... Логическое значение 30)**. Логическое значение 1; Логическое значение 2 ... Логическое значение 30 - проверяемые условия. Аргументы представляют собой либо логические выражения (TRUE, 1<5, 2+3=7, B8<10), которые возвращают логические значения, либо массивы(A1:C3) логических значений. Все условия могут иметь значение ИСТИНА (TRUE) или ЛОЖЬ (FALSE). Если в качестве параметра указан диапазон, функция использует значение из текущего столбца или строки диапазона. Примеры: =ИЛИ(A;B) возвращает значение ИСТИНА (TRUE) в случае, когда или A, или B, или и A, и B равны ИСТИНА; =ИЛИ(A;B) возвращает значение ЛОЖЬ (FALSE) в случае, когда и A, и B равны ЛОЖЬ.

Функции* И *(*AND*)*, ИЛИ *(*OR*)*, НЕ *(*NOT*) - позволяют создавать сложные логические выражения. Эти функции работают в сочетании с простыми операторами сравнения.

Аргументы функций* И, ИЛИ, НЕ *могут быть логическими выражениями, массивами или ссылками на ячейки, содержащие логические значения. Приведем пример. Пусть Excel возвращает текст □Прошел□, если ученик имеет средний балл более 4 (ячейка A2) и пропуск занятий меньше 3 (ячейка A3). Формула примет вид: **=ЕСЛИ(И(A2>4;A3<3);□Прошел□;□Не прошел□).**

Несмотря на то, что функция* ИЛИ *имеет те же аргументы, что и* И, *результаты получаются совершенно различными. Так, если в предыдущей формуле заменить функцию И на ИЛИ, то ученик будет проходить, если выполняется хотя бы одно из условий (средний балл более 4 или пропуски занятий менее 3). Таким образом, функция ИЛИ возвращает логическое значение ИСТИНА, если хотя бы одно из логических выражений истинно, а функция И возвращает логическое значение ИСТИНА, только если все логические выражения истинны.

ТЕМА 6. КОМБИНАТОРИКА

Требования: студенты должны владеть основными понятиями комбинаторики, уметь считать количество сочетаний, перестановок и размещений и просто логически мыслить.

Комбинаторный анализ, или комбинаторика, занимается изучением способов составления комбинаций из предметов. Термин □комбинаторика□ был введен в математический обиход Лейбницем.

Комбинаторика - раздел математики, изучающий дискретные объекты,

множества (сочетания, перестановки, размещения и перечисления элементов) и отношения на них (например, частичного порядка). Комбинаторика связана со многими другими областями математики — алгеброй, геометрией, теорией вероятностей, и имеет широкий спектр применения, например, в информатике и статистической физике. Иногда под комбинаторикой понимают более обширный раздел дискретной математики, включающий в частности , теорию графов. Для формулировки и решения комбинаторных задач используют различные модели комбинаторных конфигураций.

Размещением из n элементов по k называется упорядоченный набор из k различных элементов некоторого n -элементного множества. ***Перестановкой из n элементов*** (обычно чисел 1,2, … n) называется всякий упорядоченный набор из этих элементов. Перестановка также вляется размещением из n элементов по n . ***Сочетанием из n элементов по k*** называтся набор k элементов, выбранных из данных n элементов. Наборы , отличающиеся только порядком следования элементов (но не составом) , считаются одинаковыми, этим сочетания отличаются от размещений. Композицией числа n называется всякое представление n в виде упорядоченной суммы целых положительных чисел. Разбиением числа n называется всякое представление n в виде неупорядоченной суммы целых положительных чисел.

Основные сведения из комбинаторики.

Рассмотрим множество S, состоящее из n>=1 различных объектов: a_1, a_2,...,a_{n-1},a_n. Термин ***«упорядоченная группа»*** или ***«размещение» по k объектов из n объектов***, образующих S, применяется для обозначения группы k произвольных объектов, выбранных из S и расположенных в специальном порядке, так что один конкретный объект группы принимается за первый, другой — за второй, еще один — за третий и т. д. Таким образом, (a_1, a_2) и (a_1, a_5), так же как (a_2, a_1), представляют собой различные размещения двух объектов, выбранных из множества S. Две группы по k объектов из n рассматриваются как различные размещения, когда они отличаются или, по крайней мере, одним объектом, присутствующим в одной группе и отсутствующим в другой, или порядком, в котором объекты расположены, или и тем и другим.

Размещение из n элементов по k - всякая упорядоченная часть множества, содержащая k элементов. Представим себе какие-нибудь n=3 объектов, например, три карандаша, черный, зеленый и красный (для краткости, Ч, З и К), лежащих на столе, и рассмотрим все различные способы (i) выбора k=2 из этих карандашей и (j) упорядочения выбранных групп. Легко видеть, что имеется три различных способа выбора двух карандашей из заданных трех. Эти способы следующие: выбрать Ч и З и оставить К, выбрать Ч и К и оставить З, выбрать З и К и оставить Ч ; i=1,2,3. Далее,

очевидно, что каждую из групп в два объекта можно упорядочить только двумя различными путями (j=1,2). Таким образом, все различные размещения по два карандаша из трех можно записать в виде: (Ч,З), (З,Ч); (Ч,К), (К,Ч); (З,К), (К,З) - всего 6**.** Число всех возможных размещений по k объектов из n объектов принято обозначать символом A_n^k . Только что полученный результат может быть записан в виде $A_6^2=6$.

Число размещений без повторений определяется формулой:

$$A_n^k = n!/(n-k)!.$$

Размещения без повторений отличаются друг от друга или составом элементов, или их порядком, или и тем и другим.

Например, при размещении 6 цифр на 2 костях получим следующие комбинации: { 1 2; 2 1; 1 3; 3 1; 1 4; 4 1; 1 5; 5 1; 1 6; 6 1; 2 3; 3 2; 2 4; 4 2; 2 5; 5 2; 2 6; 6 2; 3 4; 4 3; 3 5; 5 3; 3 6; 6 3; 4 5; 5 4; 4 6; 6 4; 5 6; 6 5} - всего 30 комбинаций.

Число размещений A_n^k =n!/(n-k)! из n (числа элементов в исходном множестве, из которого формируются размещения) по k (число элементов в каждом размещении) в Excel : ПЕРЕСТ(число — n ; число_ выбранных – k); в Mathcad : permut(n,k) .

Если множество S состоит из N=n*k, n>=1 различных объектов, причем каждый из объектов повторяется k раз, то рассматривают размещения с повторениями, состоящие из размещений, содержащих k одинаковых объектов, и размещений, содержащих k объектов, отличающимися или самими объектами, присутствующими в одной группе и отсутствующим в другой, или порядком, в котором объекты расположены, или и тем и другим.

Число размещений с повторениями из n элементов по k определяется формулой : n^k.

Например, при n=6, k=2 получаем $6^2=36$ размещений с повторениями: к перечисленным выше 30 размещениям добавляются 6 с одинаковыми цифрами: { 1 1 ; 2 2; 3 3; 4 4; 5 5; 6 6}. Число возможных размещений с повторениями при бросании 3 костей равно $6^3=216$, при бросании 4 костей равно $6^4=1296$, при бросании 5 костей равно $6^5=7776$.

Термин ***«перестановка» из n объектов*** применяется для обозначения специального порядка, в котором эти объекты расположены, при этом один конкретный объект группы принимается за первый, другой — за второй и т. д. Например, рассмотренные три карандаша (Ч,З, К) можно расположить следующими шестью различными способами: (Ч,З,К), (Ч,К,З); (З,Ч,К), (З,К,Ч); (К,Ч.З), (К,З,Ч). Число всех возможных перестановок из n объектов принято обозначать символом P_n. Только что полученный результат может быть записан в виде $P_3=6$.

Число возможных перестановок без повторений определяется формулой Pn=n! . Перестановки без повторений отличаются друг от

друга порядком входящих в них элементов.

В примере с тремя карандашами имели: $P_3=3!=6$. Из 4 элементов можно образовать $P_4=4!=24$ перестановки; из 5 элементов можно образовать $P_5=5!=120$ перестановок; из 6 элементов можно образовать $P_6=6!=720$ перестановок.

Число перестановок с повторениями, в которых элемент a повторяется k_1 раз, элемент b повторяется k_2 раз, элемент c повторяется k_3 раз, определяется формулой $(k_1+k_2+k_3)!/(k_1!*k_2!*k_3!)$.

Например, из элементов (1; 2; 3) при $k_1=2$, $k_2=2$, $k_3=1$ (две 1, две 2, одна 3 - всего 5) можно сформировать (2+2+1)!/{2!*2!*1!}=5!//{2!*2!*1!}=30 перестановок с повторениями: {1 1 2 2 3; 1 1 2 3 2; 1 1 3 2 2; 1 2 1 2 3; 1 2 1 3 2; 1 2 3 1 2; 1 2 3 2 1; 1 2 2 1 3; 1 2 2 3 1; 1 3 2 1 2 ; 1 3 1 2 2; 1 3 2 2 1; 2 1 1 2 3; 2 1 1 3 2; 2 1 2 1 3; 2 1 2 3 1; 2 1 3 1 2; 2 1 3 2 1; 2 3 1 1 2; 2 3 1 2 1; 2 3 2 1 1; 2 2 1 1 3; 2 2 1 3 1; 2 2 3 1 1 ; 3 1 1 2 2; 3 1 2 1 2; 3 1 2 2 1; 3 2 1 1 2; 3 2 1 2 1; 3 2 2 1 1}. Если бы все 5 элементов были различными, имели бы $P_5=5!=120$.

Число перестановок в Excel : ПЕРЕСТ(число — n ; число_ выбранных – k); в Mathcad : permut(n,k) .

Термин ***«неупорядоченная группа»,*** или ***«сочетание»*** по k из n объектов, образующих множество S, применяется для обозначения группы из k произвольных объектов, взятых из S , независимо от порядка, в котором эти объекты могут быть размещены внутри самой группы. Из этого определения следует, что две группы с одинаковым числом объектов рассматриваются как различные сочетания только тогда, когда эти группы отличаются хотя бы одним объектом.

Таким образом, возвращаясь к примеру с тремя карандашами, можно заметить, что (Ч,З) и (З,Ч) являются двумя различными размещениями по два карандаша из трех карандашей, но вместе с тем они представляют собой одну и ту же неупорядоченную группу (одно и то же сочетание) двух карандашей из заданных трех. Следовательно, для этого примера число сочетаний из трех карандашей разного цвета по два равно 3.

Число возможных сочетаний по k объектов из n объектов принято обозначать символом C_n^k. ***Число возможных сочетаний определяется формулой: C_n^k= n!/[k!*(n-k)!]. Сочетания без повторений отличаются друг от друга составом входящих в них элементов.***

В примере с 3 карандашами имели: $C_3^2= 3!/(2!*(3-2)!)=3$.

Например, на 2 костях можно получить : {1 2;1 3;1 4;1 5;1 6;2 3; 2 4;2 5; 2 6;3 4 ;3 5;3 6; 4 5;4 6;5 6} - 15 комбинаций: $C_6^2= 6!/[2!* (6-2)!]=15$.

Число сочетаний C_n^k =n!/[k!*(n-k)!] из n (числа элементов в исходном множестве,из которого формируются сочетания) по k (число

элементов в каждом сочетании) в Excel : ЧИСЛКОМБ (число — n ; число_выбранных - k) ; в Mathcad : combim (n,k).

ТЕМА 7. АЛГОРИТМЫ. ПРОЦЕСС ВЫЧИСЛЕНИЙ В ЭВМ

Требования: студенты должны знать историю появления и развития теории алгоритмизации, уметь составлять алгоритмы поставленных задач и реализовать их в различных инструментальных средах.

История термина «алгоритм».

Слово «алгоритм» происходит от имени великого среднеазиатского ученого VIII-IX вв. Абу Абдуллах Мухаммеда ибн Муса аль-Хорезми. (Хорезм— историческая область на территории современного Узбекистана). Из математических работ аль-Хорезми до нас дошли только две — алгебраическая (от названия этой книги родилось слово алгебра) и арифметическая.

На протяжении многих веков происхождению слова «алгоритм» давались самые разные объяснения. Одни выводили algorism из греческих algiros (больной) и arithmos (число). Из такого объяснения не очень ясно, почему числа именно «больные». Или же лингвистам больными казались люди, имеющие несчастье заниматься вычислениями? Своё объяснение предлагал и энциклопедический словарь Брокгауза и Ефрона. В нём алгорифм (кстати, до революции использовалось написание алгори□м, через фиту) производится «от арабского слова аль-Горетм, то есть корень». Разумеется, эти объяснения вряд ли можно счесть убедительными. Перевод сочинений аль-Хорезми стал первой ласточкой, и в течение нескольких последующих столетий появилось множество других трудов, посвящённых всё тому же вопросу— обучению искусству счёта с помощью цифр. И все они в названии имели слово algoritmi или algorismi.

Однако со временем такие объяснения всё менее занимали математиков, и слово algorism (или algorismus), неизменно присутствовавшее в названиях математических сочинений, обрело значение способа выполнения арифметических действий посредством арабских цифр, то есть на бумаге, без использования абака. Абак (греч. αβαξ, abakion, лат. abacus — доска) — счётная доска, применявшаяся для арифметических вычислений.

Постепенно значение слова расширялось. Учёные начинали применять его не только к сугубо вычислительным, но и к другим математическим процедурам. Например, около 1360 г. французский философ Никола Орем (Nicolaus Oresme, 1323(25)-1382) написал математический трактат «Algorismus proportionum» («Вычисление пропорций»), в котором впервые использовал степени с дробными показателями и фактически вплотную подошёл к идее логарифмов.

Итак, сочинения по искусству счёта назывались «Алгоритмами».

Именно в таком значении это слово вошло во многие европейские языки. Например, с пометкой «устар.» оно присутствует в представительном словаре английского языка Webster's New World Dictionary, изданном в 1957 г. Алгоритм— это искусство счёта с помощью цифр, но поначалу слово «цифра» относилось только к нулю. В 1684 году Готфрид Лейбниц в сочинении «Nova Methojdys pro maximus et minimus, itemque tangentibus...» впервые использовал слово «алгоритм» (Algorithmo) в ещё более широком смысле: как систематический способ решения проблем дифференциального исчисления.

Однако потребовалось ещё почти два столетия, чтобы все старинные значения слова вышли из употребления. Этот процесс можно проследить на примере проникновения слова «алгоритм» в русский язык. Историки датируют 1691 годом один из списков древнерусского учебника арифметики, известного как «Счётная мудрость». Это сочинение известно во многих вариантах (самые ранние из них почти на сто лет старше) и восходит к ещё более древним рукописям XVI в. По ним можно проследить, как знание арабских цифр и правил действий с ними постепенно распространялось на Руси. Полное название этого учебника — «Сия книга, глаголемая по еллински и по гречески арифметика, а по немецки алгоризма, а по русски цифирная счётная мудрость».Таким образом, слово «алгоритм» понималось первыми русскими математиками так же, как и в Западной Европе. Однако его не было ни в знаменитом словаре В.И.Даля, ни спустя сто лет в «Толковом словаре русского языка» под редакцией Д.Н.Ушакова (1935 г.). Зато слово «алгорифм» можно найти и в популярном дореволюционном Энциклопедическом словаре братьев Гранат, и в первом издании «Большой Советской Энциклопедии (БСЭ)», изданном в 1926 г. И там, и там оно трактуется одинаково: как правило, по которому выполняется то или иное из четырёх арифметических действий в десятичной системе счисления. Однако к началу XX в. для математиков слово «алгоритм» уже означало любой арифметический или алгебраический процесс, выполняемый по строго определённым правилам, и это объяснение также даётся в БСЭ.

В повседневной жизни каждый человек сталкивается с необходимостью решения задач самой разной сложности. Некоторые из них трудны и требуют длительных размышлений для поиска решений (а иногда его так и не удается найти), другие же, напротив, столь просты и привычны, что решаются автоматически. При этом выполнение даже самой простой задачи осуществляется в несколько последовательных этапов (шагов). В виде последовательности шагов можно описать процесс решения многих задач, известных из школьного курса математики: приведение дробей к общему знаменателю, решение системы линейных уравнений путем последовательного исключения неизвестных, построение треугольника по трем сторонам с помощью циркуля и линейки и т.д. Такая

последовательность шагов в решении задачи называется алгоритмом. Каждое отдельное действие — это шаг алгоритма. Последовательность шагов алгоритма строго фиксирована, т. е. шаги должны быть упорядоченными.

Понятие алгоритма близко к другим понятиям, таким, как метод (метод Гаусса решения систем линейных уравнений), способ (способ построения треугольника по трем сторонам с помощью циркуля и линейки).

Алгоритмы становились предметом все более пристального внимания учёных, и постепенно это понятие заняло одно из центральных мест в современной математике. Одновременно с развитием понятия алгоритма постепенно происходила и его экспансия из чистой математики в другие сферы. И начало ей положило появление компьютеров, благодаря которому слово «алгоритм» вошло в 1985 г. во все школьные учебники «Информатики» и обрело новую жизнь. Вообще можно сказать, что его сегодняшняя известность напрямую связана со степенью распространения компьютеров. Уже в начале 70-х гг. прошлого столетия, когда компьютеры перестали быть экзотической диковинкой, слово «алгоритм» стремительно входит в обиход. В «Энциклопедии кибернетики» (1974 г.) в статье «Алгоритм» он уже связывается с реализацией на вычислительных машинах, а в «Советской военной энциклопедии» (1976 г.) даже появляется отдельная статья «Алгоритм решения задачи на ЭВМ». За последние полтора-два десятилетия компьютер стал неотъемлемым атрибутом нашей жизни, компьютерная лексика становится все более привычной.

В старой трактовке ***алгоритм*** — это точный набор инструкций, описывающих последовательность действий исполнителя для достижения результата решения задачи за конечное время. По мере развития параллельности в работе компьютеров слово «последовательность» стали заменять более общим словом «порядок». Это связано с тем, что какие-то действия алгоритма должны быть выполнены только друг за другом, но какие-то могут быть и независимыми. Часто в качестве исполнителя выступает некоторый механизм (компьютер, токарный станок, швейная машина), но понятие алгоритма необязательно относится к компьютерным программам, так, например, чётко описанный рецепт приготовления блюда также является алгоритмом, в таком случае исполнителем является человек.

Единого, «истинного» определения понятия «алгоритм» нет. Приведем некоторые из них.

«Алгоритм - это конечный набор правил, который определяет последовательность операций для решения конкретного множества задач и обладает пятью важными чертами: конечность, определённость, ввод, вывод, эффективность» (Д.Э.Кнут). «Алгоритм— это всякая система вычислений, выполняемых по строго определённым правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи»
(А.Н. Колмогоров). «Алгоритм— это точное предписание, определяющее

вычислительный процесс, идущий от варьируемых исходных данных к искомому результату» (А.А. Марков). «Алгоритм— точное предписание о выполнении в определенном порядке некоторой системы операций, ведущих к решению всех задач данного типа» (Философский словарь / Под ред. М.М. Розенталя). «Алгоритм— строго детерминированная последовательность действий, описывающая процесс преобразования объекта из начального состояния в конечное, записанная с помощью понятных исполнителю команд» (Н. Д. Угринович, учебник «Информатика и информационные технологии»). «Алгоритм— это последовательность действий, направленных на получение определённого результата за конечное число шагов» (Е.Н. Привалов). «Алгоритм— однозначно, доступно и кратко (условные понятия— названия этапа) описанная последовательность процедур для воспроизводства процесса с обусловленным задачей алгоритма результатом при заданных начальных условиях. Универсальность (или специализация) алгоритма определяется применимостью и надёжностью данного алгоритма для решения нестандартных задач». «Алгоритм— это понятные и точные предписания исполнителю совершить конечное число шагов, направленных на решение поставленной задачи». «Алгоритм— это некоторый конечный набор рассчитанных на определённого исполнителя операций, в результате выполнения которых через определённое число шагов может быть достигнута поставленная цель или решена задача определённого типа». «Алгоритм— это последовательность действий, либо приводящая к решению задачи, либо поясняющая, почему это решение получить нельзя». «Алгоритм - это точная, однозначная, конечная последовательность действий, которую должен выполнить пользователь для достижения конкретной цели либо для решения конкретной задачи или группы задач». «Алгоритм - это точное предписание, которое задает вычислительный (алгоритмический) процесс, начинающийся с произвольного исходного данного и направленный на получение полностью определяемым этим исходным данным результата». «Алгоритм— любая информация, для которой существует интерпретатор».

Можно сформулировать основные особенности именно алгоритмов, исходя из всех вышеприведенных определений.

1) ***Наличие исходных данных и некоторого результата***.

Алгоритм - это точно определенная инструкция, последовательно применяя которую к исходным данным, можно получить решение задачи. Для каждого алгоритма есть некоторое множество объектов, допустимых в качестве исходных данных. Например, в алгоритме деления вещественных чисел делимое может быть любым, а делитель не может быть равен нулю.

2) ***Массовость***.

Возможность применять многократно один и тот же алгоритм. Алгоритм служит, как правило, для решения не одной конкретной задачи, а некоторого класса задач. Так, алгоритм сложения применим к любой паре

натуральных чисел.

3) ***Детерминированность***.

При применении алгоритма к одним и тем же исходным данным должен получаться всегда один и тот же результат, поэтому, например, процесс преобразования информации, в котором участвует бросание монеты, не является детерминированным и не может быть назван алгоритмом.

4) ***Результативность***.

Выполнение алгоритма должно обязательно приводить к его завершению. В то же время, можно привести примеры формально бесконечных алгоритмов, широко применяемых на практике. Например, алгоритм работы системы сбора метеорологических данных состоит в непрерывном повторении последовательности действий («измерить температуру воздуха», «определить атмосферное давление»), выполняемых с определенной частотой (через минуту, час) во все время существования данной системы.

5) ***Определенность.***

На каждом шаге алгоритма у исполнителя должно быть достаточно информации, чтобы его выполнить. Кроме того, исполнителю нужно четко знать, каким образом он выполняется. Шаги инструкции должны быть достаточно простыми, элементарными, а исполнитель должен однозначно понимать смысл каждого шага последовательности действий, составляющих алгоритм (при вычислении площади прямоугольника любому исполнителю нужно уметь умножать и трактовать знак «*» именно как умножение). Поэтому вопрос о выборе формы представления алгоритма очень важен. Фактически речь идет о том, на каком языке записан алгоритм.

Формы представления алгоритмов.

Для записи алгоритмов необходим некоторый язык, при этом очень важно, какой именно язык выбран. Записывать алгоритмы на русском языке (или любом другом естественном языке) громоздко и неудобно.

Например, описание алгоритма Евклида нахождения НОД (наибольшего общего делителя) двух целых положительных чисел может быть представлено в виде трех шагов.

Шаг 1: Разделить m на n. Пусть p - остаток от деления.

Шаг 2: Если p равно нулю, то n и есть исходный НОД.

Шаг 3: Если p не равно нулю, то сделаем m равным n, а n равным p.
Вернуться к шагу 1.

Приведенная здесь запись алгоритма нахождения НОД очень упрощенная. Запись, данная Евклидом, представляет собой страницу текста, причем последовательность действий существенно сложней.

Одним из распространенных способов записи алгоритмов является запись на языке блок-схем. Запись представляет собой набор элементов (блоков), соединенных стрелками. Каждый элемент — это «шаг» алгоритма.

Элементы блок-схемы делятся на два вида. Элементы, содержащие инструкцию выполнения какого-либо действия, обозначают прямоугольниками, а элементы, содержащие проверку условия— ромбами. Из прямоугольников всегда выходит только одна стрелка (входить может несколько), а из ромбов — две (одна из них помечается словом «да», другая - словом «нет», они показывают, соответственно, выполнено или нет проверяемое условие). Построение блок-схем из элементов всего лишь нескольких типов дает возможность преобразовать их в компьютерные программы и позволяет формализовать этот процесс.

Формализация понятия алгоритмов.

Математические определения фигур, чисел, уравнений, неравенств и многих других объектов очень четки. Каждый математически определенный объект можно сравнить с другим объектом, соответствующим тому же определению. Например, прямоугольник можно сравнить с другим прямоугольником по площади или по длине периметра. Возможность сравнения математически определенных объектов — важный момент математического изучения этих объектов. Рассмотренные выше определения алгоритма не позволяют сравнивать какие-либо две таким образом определенные инструкции. Можно, например, сравнить два алгоритма решения системы уравнений и выбрать более подходящий в данном случае, но невозможно сравнить алгоритм перехода через улицу с алгоритмом извлечения квадратного корня. С этой целью нужно формализовать понятие алгоритма, т.е. отвлечься от существа решаемой данным алгоритмом задачи, и выделить свойства различных алгоритмов, привлекая к рассмотрению только форму их записи.

Задача нахождения единообразной формы записи алгоритмов, решающих различные задачи, является одной из основных задач теории алгоритмов.

В теории алгоритмов предполагается, что каждый шаг алгоритма таков, что его может выполнить достаточно простое устройство (машина). Желательно, чтобы это устройство было универсальным, т.е. чтобы на нем можно было выполнять любой алгоритм. Механизм работы машины должен быть максимально простым по логической структуре, но настолько точным, чтобы эта структура могла служить предметом математического исследования. Впервые это было сделано американским математиком Эмилем Постом в 1936 году (машина Поста) еще до создания современных вычислительных машин и (практически одновременно) английским математиком Аланом Тьюрингом (машина Тьюринга). Машина Поста - абстрактная вычислительная машина, предложенная Постом, отличается от машины Тьюринга большей простотой. Обе машины «эквивалентны» и были созданы для уточнения понятия «алгоритм».

Современный взгляд на алгоритмизацию.

Теория алгоритмов строит и изучает конкретные модели алгоритмов. С развитием вычислительной техники и теории программирования возрастает необходимость построения новых экономичных алгоритмов, изменяются способы их построения, способы записи алгоритмов на языке, понятном исполнителю. Особый тип исполнителя алгоритмов — компьютер, поэтому необходимо создавать специальные средства, позволяющие, с одной стороны, разработчику в удобном виде записывать алгоритмы, а с другой — дающие компьютеру возможность понимать написанное. Такими средствами являются ***языки программирования, или алгоритмические языки***.

Ещё раз сформулируем и уточним следующий ряд общих требований, предъявляемых к алгоритмам:

1) ***детерминированность - определённость***. В каждый момент времени следующий шаг работы однозначно определяется состоянием системы. Таким образом, алгоритм выдаёт один и тот же результат (ответ) для одних и тех же исходных данных. В современной трактовке у разных реализаций одного и того же алгоритма должен быть изоморфный граф. С другой стороны, существуют вероятностные алгоритмы, в которых следующий шаг работы зависит от текущего состояния системы и генерируемого случайного числа. Однако при включении метода генерации случайных чисел в список «исходных данных», вероятностный алгоритм становится подвидом обычного;

2) ***понятность*** - алгоритм для исполнителя должен включать только те команды, которые ему (исполнителю) доступны, которые входят в его систему команд;

3) ***завершаемость*** (конечность) - при корректно заданных исходных данных алгоритм должен завершать работу и выдавать результат за конечное число шагов. С другой стороны, вероятностный алгоритм может и никогда не выдать результат, но вероятность этого равна 0;

4) ***массовость*** - алгоритм должен быть применим к разным наборам исходных данных;

5) ***результативность*** - завершение алгоритма определенными результатами;

6) ***алгоритм содержит ошибки***, если приводит к получению неправильных результатов либо не дает результатов вовсе;

7) ***алгоритм не содержит ошибок***, если он дает правильные результаты для любых допустимых исходных данных .

Алгоритм— это точно определённая инструкция, последовательно применяя которую к исходным данным, можно получить решение задачи. Для каждого алгоритма есть некоторое множество объектов, допустимых в качестве исходных данных.

Например, в алгоритме деления вещественных чисел делимое может быть любым, а делитель не может быть равен нулю.

Алгоритм служит, как правило, для решения не одной конкретной задачи, а некоторого класса задач.

Так, алгоритм сложения применим к любой паре натуральных чисел (в этом выражается его свойство массовости).

Виды алгоритмов.

Особую роль выполняют ***прикладные алгоритмы***, предназначенные для решения определенных прикладных задач. Алгоритм считается правильным, если он отвечает требованиям задачи (например, даёт физически правдоподобный результат). Алгоритм (программа) содержит ошибки, если для некоторых исходных данных он дает неправильные результаты, сбои, отказы или не дает никаких результатов вообще. Важную роль играют ***рекурсивные алгоритмы*** (алгоритмы, вызывающие сами себя до тех пор, пока не будет достигнуто некоторое условие возвращения). Начиная с конца XX века активно разрабатываются ***параллельные алгоритмы***, предназначенные для вычислительных машин, способных выполнять несколько операций одновременно.

Для разработки алгоритмов и программ используется ***алгоритмизация*** **-** процесс систематического составления алгоритмов для решения поставленных прикладных задач. Алгоритмизация считается обязательным этапом в процессе разработки программ и решения задач на ЭВМ. Именно для прикладных алгоритмов и программ принципиально важны детерминированность, результативность и массовость, а также правильность результатов решения поставленных задач.

Форма записи алгоритмов.

Алгоритм может быть записан словами и изображён схематически. Обычно сначала (на уровне идеи) алгоритм описывается словами, но по мере приближения к реализации он обретает всё более формальные очертания и формулировку на языке, понятном исполнителю (например, машинный код). Например, для описания алгоритма применяются блок-схемы . Другим вариантом описания, не зависимым от языка программирования, является псевдокод.

Эффективность алгоритмов.

Хотя в определении алгоритма требуется лишь конечность числа шагов, необходимых для достижения результата, на практике выполнение даже хотя бы миллиарда шагов является слишком медленным. Также обычно есть другие ограничения (на размер программы, на допустимые действия). В связи с этим вводят такие понятия как сложность алгоритма (временная, по размеру программы, вычислительная и др.). Для каждой задачи может существовать множество алгоритмов, приводящих к цели. Увеличение эффективности алгоритмов составляет одну из задач современной информатики. В 50-х годах XX века появилась даже отдельная её область— быстрые алгоритмы. В частности, в известной всем с детства задаче об умножении десятичных

чисел обнаружился ряд алгоритмов, позволяющих существенно (в асимптотическом смысле) ускорить нахождение произведения.

Вычисления - многошаговый процесс, состоящий из последовательности действий. И перед тем, как приступить к вычислениям, ***нужно знать метод решения задачи***, т. е. располагать предписанием, в котором указано, какие действия и в каком порядке следует выполнять, чтобы решить задачу. Такого рода предписание, определяющее порядок вычислений над данными с целью получения искомых результатов, называется ***алгоритмом***. ***Алгоритм - система правил, сформулированная на понятном исполнителю языке, которая определяет процесс перехода от допустимых исходных данных к некоторому результату и обладает свойствами массовости, конечности, определенности, детерминированности.***

Определив максимально конкретно формулировку понятия «алгоритм» и основные его свойства, поставим несколько вопросов, ответы на которые помогут нам наиболее эффективно работать с алгоритмами, составляемыми для решения конкретных задач.

Как записать алгоритм?

Пусть задана последовательность из n чисел $x_1,x_2,...,x_n$, требуется определить сумму положительных и сумму отрицательных членов этой последовательности. Введем обозначения величин, которые будут использоваться в алгоритме. При решении задачи придется поочередно обращаться к членам последовательности. Отдельный член последовательности будем обозначать как x_i, т. е выделять его с помощью номера i, называемого индексом. Индекс i может принимать только целые значения 1,2,...,n. Сумму положительных членов обозначим P, а сумму отрицательных членов Q . Итак, на момент начала вычислений заданы: количество данных n и сами данные $\{x_1,x_2,...,x_n\}$, т. е. (n+1) значений. Исходя из них, нужно вычислить значения величин P и Q.

Алгоритм состоит из последовательности действий. Отдельные действия будем обозначать числами 1,2,3, ... Эти числа имеют смысл меток действий. С использованием введенных обозначений величин и действий правило решения поставленной задачи можно описать следующим образом:

1. Положить P=0,Q=0.
2. Положить индекс i=1.
3. Проверить знак x_i : если $x_i > 0$, то перейти к действию 4, иначе к действию 6.
4. Добавить к P значение x_i: $P=P+x_i$.
5. Перейти к действию 7.
6. Добавить к Q значение x_i: $Q=Q+x_i$.
7. Если $i<n$, то перейти к действию 8, иначе закончить вычисления.

8. Вычислить новое значение индекса i=i+1.

9. Перейти к действию 3.

Эта последовательность из девяти действий и представляет собой алгоритм решения поставленной задачи. Прокомментируем смысл отдельных действий. В процессе вычислений последовательно наращивались значения переменных P и Q путем добавления к ним сначала значения x_1, затем значения x_2 и т. д. Следовательно, перед началом этого процесса нужно значения сумм P и Q положить равными нулю (действие 1).

Теперь рассмотрим выражение $P=P+x_i$, используемое в действии 4. С математической точки зрения это выражение бессмысленно, так же как бессмысленно выражение i=i+1, поскольку равенство невозможно ни при каком значении i. Однако в рамках алгоритмов такие выражения понимаются не в традиционном смысле — не как равенства, а следующим образом: выражение $P=P+x_i$ означает , что вычисляется сумма $P+x_i$, значение которой присваивается величине P . Здесь подразумевается, что в левой и правой частях выражения $P=P+x_i$ используются разные значения переменной P. В правой части используется старое значение P, а в левой - новое значение P, увеличенное на x_i .

Теперь рассмотрим, в чем назначение действия 5. При описании алгоритмов предполагается, что действия выполняются в естественном порядке — в порядке их перечисления: сначала выполняется действие 1, затем действие 2, а затем действие 3. Однако после выполнения действия 4 этот порядок должен быть непременно нарушен, поскольку недопустимо вслед за действием 4 начать действие $Q=Q+x_i$. Чтобы нарушить естественный порядок выполнения действий, и служит действие 5, переводящее процесс вычислений к действию 7, минуя действие 6. Построенному алгоритму присущи все необходимые свойства: порядок решения задачи определен с необходимой строгостью; значения результатов вычисляются для любого конечного n=>1 (заметим, что при n=0 , т.е. при отсутствии членов в последовательности, данный алгоритм не работает. Случай n=0, конечно, можно учесть. Для этого алгоритм нужно изменить так, чтобы при n=0 обеспечить получение нулевых сумм P=0, Q=0. Это можно сделать самостоятельно.) ; результаты получены за конечное число шагов вычислений.

Из чего строятся алгоритмы?

Из приведенного выше примера можно составить представление о том, из чего и как строятся алгоритмы. Отдельные действия, на которые расчленяется алгоритм, принято называть ***операторами алгоритма***.

В алгоритме используются операторы двух типов.

К ***первому типу*** относятся операторы, обеспечивающие вычисление значений величин. Такие операторы можно рассматривать как ***основные***

операторы (это операторы 1, 2, 4, 6 и 8 рассмотренного выше алгоритма). Основные операторы строятся из ***арифметических операций*** и ***операций присваивания***. Операция присваивания обеспечивает перенесение значения на величину и обозначается знаком « = », который в рассмотренном алгоритме означает «положить равным». Указанные операции распространяются на переменные величины (в приведенном примере на переменные $P,Q,x_1,...,x_n$, i и n) и константы, представляемые числами. Вообще говоря, кроме арифметических операций в алгоритмах используются ***логические операции*** для выполнения действий над логическими значениями «истина» и «ложь», а также специальные операции над строками символов.

Ко ***второму типу*** операторов, используемых в алгоритмах, относятся операторы , задающие порядок выполнения основных операторов (это операторы 5, 7 и 9 рассмотренного выше алгоритма). Операторы, управляющие порядком выполнения действий, называются ***операторами перехода***. Операторы этого типа могут предписывать ***переход безусловно***. Таков в примере смысл операторов 5 и 9. В отличие, от этого операторы 3 и 7 анализируют ситуацию и в зависимости от того, имеет место эта ситуация или нет, обеспечивают переход либо в одном, либо в другом направлении к разным операторам, но всегда только к одному из них. В таких операторах перехода используется специальное выражение, называемое ***распознавателем***, а чаще ***предикатом***. Так, в операторах 3 и 7 используются предикаты $x_i > 0$ и $i<n$. Предикат может принимать одно из двух значений: «истина» или «ложь». При n=100 во всех случаях, когда индекс i принимает значения от 1 до 99 включительно, предикат $i<n$ имеет значение «истина», но когда i=100, этот предикат принимает значение «ложь». ***Оператор условного перехода*** в зависимости от истинности и ложности условия, т. е. предиката, направляет процесс вычислений либо к одному, либо к другому оператору. Таким образом, алгоритм — совокупность операторов, часть которых - основные операторы описывают необходимые действия над информацией, а другая часть - операторы перехода задают порядок выполнения действий.

Как выполняются алгоритмы?

Алгоритм — это только правило решения задачи. Само же решение , т. е. значения искомых результатов, получают выполнением алгоритмов. ***Выполнить алгоритм*** - значит выполнить действия, предписанные операторами алгоритма.

Вот как будет выглядеть процесс выполнения описанного выше алгоритма в случае, когда последовательность {x1,x2,...,xn } содержит **n=5** членов со значениями **(-3);2;5;(-1);4.**

Шаг 1. Оператор 1. **P=0,Q=0.**

Шаг 2. Оператор 2. **i=1.**

Шаг 3. Оператор 3. -3>0? Нет. Переход к оператору 6.

Шаг 4. Оператор 6. **Q=Q +x1=0+(-3)=-3.**
Шаг 5. Оператор 7. 1≤ 5? Да. Переход к оператору 8.
Шаг 6. Оператор 8. **i=i+1=1+1=2.**
*Шаг 7.*Оператор 9. Переход к оператору 3.
*Шаг 8.*Оператор 3. 2>0? Да. Переход к оператору 4.
Шаг 9. Оператор 4. **P=P +x2=0+2=2.**
Шаг 10. Оператор 7. 2≤ 5? Да. Переход к оператору 8.
Шаг 11. Оператор 8**.** ***i=i+1=2+1=3.***
Шаг 12. Оператор *9.* Переход к оператору 3.
Шаг 13. Оператор 3. **5>0? Да. Переход к оператору 4.**
Шаг 14. Оператор 4. **P=P +x3=2+5=7.**
Шаг 15. Оператор 7. *3≤ 5? Да. Переход к оператору 8.*
Шаг 16. Оператор 8. ***i=i+1=3+1=4.***
Шаг 17. Оператор *9.* Переход к оператору 3.
Шаг 18. Оператор 3. -1>0? Нет. Переход к оператору 6.
Шаг 19. Оператор 6. **Q=Q +x4=-3+(-1)=-4.**
Шаг 20. Оператор 7. 4≤ 5? Да. Переход к оператору 8.
Шаг 21. **Оператор 8. i=i+1=4+1=5.**
Шаг 22. Оператор 9. Переход к оператору 3.
Шаг 23. **Оператор 3. 4>0? Да. Переход к оператору 4.**
Шаг 24. Оператор 4. **P=P +x5=7+4=11.**
Шаг 25. ***Оператор 7. 5≤ 5? Нет.***

Вычисления завершены: P= 11; Q=-4.

Такого рода процесс, порождаемый алгоритмом, называется ***вычислительным процессом*** — последовательностью действий, предписанных операторами алгоритма. Число действий, выполняемых в процессе реализации алгоритма, в общем случае не совпадает с числом операторов и обычно значительно больше числа операторов (в примере имеем 25 шагов при 9 операторах). Если бы рассматривалась последовательность из 100 чисел, различие между числом шагов (не менее 500) и числом операторов (9) было бы существенно большим. Это значит, что десяток операторов может порождать при одной реализации алгоритма сотни, тысячи и миллионы действий.

Из чего складывается процесс вычисления?

Процесс решения задачи на ЭВМ должен протекать в какой-то степени аналогично процессу решения задачи, выполняемой человеком вручную. Любые вычисления возможны только тогда, когда метод решения задачи известен, т. е. определен в форме алгоритма. При вычислениях на калькуляторе (вручную) алгоритм надо строить с использованием только простейших операций (арифметических, вида «если x≥y, то», «перейти к»).

Как свести вычисления к выполнению простейших операций?

Примем, что алгоритм в нужной форме задан — он написан на одном или нескольких листах бумаги (как в рассмотренном выше примере). Исходные данные для решения задачи также даны на отдельных листах. В процессе вычислений появятся сначала промежуточные значения, а затем конечные результаты. Их будем фиксировать на листах бумаги, которые для удобства могут быть определенным образом разграфлены. Теперь можно приступать к вычислениям. Читаем первую фразу алгоритма — это первый оператор, первое указание к действию, определяющее, что и над чем нужно делать. Выполнение оператора алгоритма сводится к следующему: нужно отыскать на листах бумаги необходимые величины — исходные данные или промежуточные значения, считать их с листа бумаги и перенести на клавиатуру калькулятора, затем нажать кнопку со знаком операции, заданной в операторе. Результат операции, вычисленный калькулятором, необходимо записать на лист, если он не будет использован в качестве аргумента в следующей операции. Далее потребуется вновь обращение к алгоритму. Следующий оператор всегда обозначает либо действие, для которого потребуется калькулятор, либо номер оператора, к которому нужно перейти, чтобы продолжить вычисления. И так до тех пор, пока не будет выполнен оператор «закончить вычисления», которым завершается любой алгоритм.

Можно ли формализовать процесс вычислений?

Весь процесс вычисления прост, и вместе с тем его детали очень важны, так как речь идет об автоматизации вычислений. Важно отметить следующее:

1) В процессе вычислений необходимо располагать записью алгоритма и средством, на котором будут фиксироваться значения: исходные данные, промежуточные и конечные результаты.

2) Необходимо располагать средствами для реализации операторов алгоритма. Такими при ручном счете являлись калькулятор и сам человек, который следил за порядком выполнения операторов, осуществлял переход от одного оператора к другому и инициировал операции на клавишной машине. Человек всегда помнил, какой оператор сейчас выполняется, и благодаря этому мог определить , каким будет следующий оператор.

3) Каждый очередной оператор алгоритма реализуется в принципе одинаково: выборка исходных значений, так называемых операндов, выполнение операции (арифметической или перехода) и фиксация результата операции (запись вычисленного значения или переход к очередному оператору).

Нужно ли мыслить при такой организации вычислений на основе алгоритма?

Положим, что алгоритм составлен идеально, тогда в процессе вычислений вообще не нужно мыслить — не нужно творить. Человек должен выполнять лишь простейшие механические действия, ничуть не сложнее

действий, выполняемых калькулятором, наличие каких бы то ни было мыслительных способностей в котором отвергается. Итак, чтобы реализовать вычисления на основе алгоритма, необходимо обеспечить выполнение вполне элементарных действий. Если для этого использовать подходящие механизмы, связать эти механизмы воедино, то получится машина, выполняющая любые вычисления автоматически, без участия человека. Однако реально при решении любой задачи участие человека необходимо, так как только человек, знающий сущность решаемой задачи, может оценить «качество» полученных результатов и соответствие их «реальности». Поэтому в реальности необходимо совместное взаимодействие «человек-машина».

Где хранить информацию?

Для автоматизации вычислений необходимо заменить каким-то устройством листы бумаги, используемые для описания алгоритма и хранения величин, участвующих в вычислениях. Это устройство должно как бы помнить алгоритм, исходные данные, промежуточные и конечные результаты, т. е. должно служить машинной памятью. Эта память не имеет ничего общего с памятью человека, да и любого живого существа. Машинная память и память человека различны как по принципу построения, так и по своим возможностям. Наилучшим синонимом термина «машинная память» является термин «склад информации». В ячейке памяти может храниться одно число или оператор алгоритма. Если смысл информации, находящейся в ячейке, имеет не значения то говорят, что в ячейке хранится ***слово информации*** — число, совокупность символов текста, оператор и т. п. Ячейки нумеруются числами 0,1,2,3, называемыми ***адресами ячеек***. Если необходимо записать в память слово, следует указать адрес ячейки, в которую надо его поместить, и подать само слово на вход памяти. Память устроена так, что заданное слово будет передано в ячейку с указанным адресом и может храниться там сколь угодно долго. В любой момент, обратившись к памяти, можно получить значение хранимого там слова. Для этого в память следует послать адрес, определяющий местоположение требуемого слова, и она через какое-то время выдаст копию слова. При этом содержимое ячейки останется без изменения , так что один раз записав слово, можно сколь угодно раз получать его копии.

Как видно, принцип построения машинной памяти достаточно прост. Впрочем, обратим внимание на одну существенную деталь. В алгоритмах величины фигурируют под своими символическими именами: x, y, P , Q и т. д. Однако от этого естественного способа именования величин приходится отказаться, поскольку именем может выступать только адрес, определяющий номер ячейки, в которой хранится значение величины. специфика машинной памяти такова, что в качестве наименования величины. Так, если величина x размещена в ячейке 275 памяти, то адрес 275 должен рассматриваться в

качестве машинного имени величины x. Таким образом, наименования величин x, y, P , Q не могут исполнять свою роль при использовании вышеописанной памяти и должны быть заменены машинными именами - адресами ячеек, в которых хранятся значения этих величин. С учетом этого, описание алгоритма должно быть изменено: общепринятые математические наименования величин, фигурирующие в алгоритме , должны быть заменены соответствующими адресами.

Как представить алгоритм машине?

Из-за специфики машинной памяти приходится представлять алгоритм в форме, допускающей его реализацию на вычислительной машине. Машинно-ориентированная форма представления алгоритма называется программой. Эта форма представления алгоритма такова. В программе операторы алгоритма представляются в виде команд. Команда - слово информации, предписывающая, как и оператор алгоритма, операцию над определенными величинами. Команда имеет следующую структуру : «код операции; адрес; адрес; адрес» . В данном случае команда состоит из четырех частей. Каждая часть команды - это группа из определенного числа символов, обычно цифр, совокупность которых и составляет слово информации. Первая часть команды определяет название операции и называется ***кодом операции***. Три последние части команды — адреса величин, участвующих в операции. Отметим, что любая команда по своей форме — это число. С этой информацией машине гораздо легче разобраться , чем с выражением. Каждая команда предписывает действие: выбрать величину из памяти, выполнить заданную операцию и записать результат в память. После того, как одна команда будет выполнена, должна выполняться другая команда. Ее местоположение - следующая ячейка памяти. Операторы алгоритма не всегда выполняются в порядке их записи. Это присуще и командам программы. Последняя может содержать команды, предписывающие переход к любым другим командам, адреса которых указаны в операторах перехода.

Что необходимо хранить в машинной памяти ?

Перед началом вычислений в память необходимо ввести программу и значения исходных данных. Значения исходных данных вводятся в память извне перед началом вычислений. В ячейки, отведенные для хранения значений промежуточных и конечных величин, будут записываться значения, полученные в результате выполнения соответствующих команд программы.

Какие средства нужны машинам для выполнения программ?

Допустим, что программа и необходимые данные загружены в память машины. Теперь необходимо выполнять вычисления, т. е. действия, заданные командами программы. При ручных вычислениях операторы алгоритма читались человеком, выполняющим действия , предписанные оператором с помощью клавишной вычислительной машины или в уме. В машине, автоматизирующей процесс вычислений, эти функции возлагаются на

процессор.

Программа может размещаться в любой области памяти. Перед началом вычислений процессору должен быть указан адрес ячейки начала программы, определяющий местоположение в памяти первой команды программы. Только после этого он может приступить к выполнению вычислений по заданной программе. При этом процессор работает следующим образом:

1. Чтение команды.
2. Дешифрирование кода операции.
3. Выборка операндов.
4. Выполнение операции.
5. Запись результата.
6. Определение адреса следующей команды.

Важно отметить, что процессор не проявляет какой-либо инициативы в своих действиях. Он делает только то, что предписано программой, предписано точно и однозначно. Программа предопределяет весь ход процесса вычислений — от начала до конца.

Как из устройства создать машину?

Для автоматического выполнения вычислений, очевидно, необходимо соеденить в одно целое процессор и память. Но этого мало. Перед началом вычислений в памяти должна присутствовать программа вычислений и исходные данные, подготавливаемые человеком. Чтобы их ввести в память машины, необходимо специальное устройство - устройство ввода. Для вывода из памяти машины результатов вычислений и представления их в форме, доступной человеку , нужно в состав машины ввести еще одно устройство - устройство вывода.

Таким образом, вычислительная машина должна состоять из следующих частей: памяти, процессора, устройства ввода и устройства вывода. Принцип автоматизации вычислений в ЭВМ состоит в организации вычислений с помощью программы, т. е. На основе последовательности команд. Именно программа настраивает ЭВМ на решение задачи, задает путь к получению искомых результатов.

На основе одного и того же принципа автоматизации вычислений — программного управления процессом вычислений можно построить разные ЭВМ, отличающиеся составом операций, выполняемых машиной, количеством информации, которую может вместить память, скоростью выполнения операций.

Все ли задачи разрешимы с помощью ЭВМ?

Поясним смысл фразы «решить задачу». Чтобы решить задачу, сначала надо придумать способ ее решения или, говоря профессионально, создать алгоритм решения задачи. И только когда известен алгоритм, можно приступать к выполнению действий, предписанных алгоритмом и приводящих к получению искомых результатов. Следовательно, решить

задачу — это разработать алгоритм плюс выполнить необходимые вычисления.

«Универсальный решатель задач», порождающий требуемый алгоритм и на основе последнего вычисляющий искомые результаты, - неосуществимая идея. Одна из причин этого — существование алгоритмически неразрешимых задач. Заметим, что алгоритмическая неразрешимость задачи вовсе не исключает возможность ее решения. Ничего парадоксального в этом нет и вот почему. Вспомним, что одно из основных свойств алгоритма - массовость. Это означает, что алгоритм представляет собой метод решения не какой-то единственной конкретной задачи, а многих задач, различающихся по своим данным. Алгоритм решения проблемы может не существовать, и в то же время каждая конкретная задача может быть решена своим методом. Однако обобщить эти частные методы так, чтобы они стали применимы к любой задаче, не всегда возможно.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Романова Ю.Д., Лесничая И.Г. Информатика и информационные технологии: учебное пособие. - 2-е изд. перераб. и доп. - М.:Экспо, 2009.

Чекмарев Ю.В. Вычислительные системы, сети и телекоммуникации, издание второе, исправленное и дополненное. - М.: ДМК Пресс, 2009.

Сафронов И.К. Готовимся к ЕГЭ. Информатика. - 2-е изд. перераб. и доп. – СПб.: БХВ - Петербург, 2009.

Степанов А.Н. Информатика: учебник для вузов. - 5-е изд. - Изд-во СПб.: Питер, 2008.

Макарова Н.В. Информатика. Практикум на ЭВМ.- СПб.: Питер, 2004.

М.А.Беляев, В.В. Лысенко, Л.А. Малинина Основы информатики: учебник для вузов.- М.: Высшее образование. Изд-во Феникс, 2006.

Симонович С.В., Евсеев Г.А. , Мураховский В.И. Информатика: базовый курс для вузов. Изд-во СПб.: Питер, 2009 г.

СОДЕРЖАНИЕ

Учебное издание

Дмитрий Георгиевич Подобед
Марианна Вячеславовна Подобед
Ольга Валентиновна Подобед

ОСНОВЫ ИНФОРМАТИКИ

(БАЗОВЫЕ МАТЕРИАЛЫ ДЛЯ КУРСА ЛЕКЦИЙ)

Учебное пособие

Редактор и корректор Н.П.Новикова
Техн. редактор Л.Я.Титова Тем. план 2010 г., поз. 36

Подп. к печати 24.03.2010 . Формат 60x86/16.Бумага тип.№ 1.
Печать офсетная. 4,75 уч.-изд. л.; 4,75 усл. печ. л.
Тираж 50 экз. Изд.№ 36. Цена «С». Заказ _____.

Ризограф ГОУВПО Санкт-Петербургского государственного технологического университета растительных полимеров, 198095, СПб., ул. Ивана Черных, 4.