«Введение в основы кодирования»

Copyright (c) 2017, Михаил Шихов (github.com/mmshihov, [m.m.shihov@gmail.com](mailto:m.m.shihov@gmail.com))

Оглавление

[1 Информация 3](#_Toc285059679)

[1.1 Определения информации 3](#_Toc285059680)

[1.2 Точки зрения на информацию 4](#_Toc285059681)

[1.3 Свойства информации 5](#_Toc285059682)

[1.4 Подходы к количественной оценке свойств информации 7](#_Toc285059683)

[1.4.1 Синтаксический подход 8](#_Toc285059684)

[1.4.2 Семантический подход 18](#_Toc285059685)

[1.4.3 Прагматический подход 19](#_Toc285059686)

[2 Кодирование 21](#_Toc285059687)

[2.1 Формальное определение кодирования 24](#_Toc285059688)

[2.2 Оптимальное кодирование 26](#_Toc285059689)

[2.3 Кодирование с целью сжатия 33](#_Toc285059690)

[2.4 Кодирование чисел 36](#_Toc285059691)

[2.4.1 Позиционные системы счисления 36](#_Toc285059692)

[2.4.2 Перевод чисел из одной системы счисления в другую 40](#_Toc285059693)

[2.4.3 Двоичная система счисления 43](#_Toc285059694)

[2.4.4 Восьмеричная и шестнадцатеричная системы счисления. 48](#_Toc285059695)

[2.4.5 Форматы чисел в вычислительных машинах 51](#_Toc285059696)

[2.5 Кодирование текста. 56](#_Toc285059697)

[2.5.1 Азбука Морзе 57](#_Toc285059698)

[2.5.2 Азбука Брайля 60](#_Toc285059699)

[2.5.3 Текст в вычислительных машинах 62](#_Toc285059700)

[2.6 Кодирование с целью защиты свойств информации 66](#_Toc285059701)

[2.6.1 Защита целостности информации 66](#_Toc285059702)

[2.6.2 Защита конфиденциальности информации 76](#_Toc285059703)

[2.6.3 Защита принадлежности информации 84](#_Toc285059704)

[2.7 Кодирование данных 88](#_Toc285059705)

[2.7.1 Кодирование идентификационных данных на примере EAN-13 90](#_Toc285059706)

[2.7.2 Текстовый и бинарный форматы данных 95](#_Toc285059707)

[2.7.3 Язык разметки XML 97](#_Toc285059708)

[2.7.4 Язык логической разметки печатных документов LaTeX 102](#_Toc285059709)

[2.7.5 Форматы графических данных 106](#_Toc285059710)

[2.7.5.1 Формат BMP 116](#_Toc285059711)

[2.7.5.2 Формат XBM 126](#_Toc285059712)

[2.7.5.3 Формат TIFF 127](#_Toc285059713)

[2.7.5.4 Формат SVG 132](#_Toc285059714)

[2.7.6 Форматы аудио данных 135](#_Toc285059715)

[2.7.6.1 Формат WAV 140](#_Toc285059716)

[2.7.7 Форматы видео данных 145](#_Toc285059717)

[2.7.7.1 Формат AVI 146](#_Toc285059718)

[2.8 Кодирование действий 151](#_Toc285059719)

[2.8.1 Регулярные выражения: поиск элементов регулярных множеств 153](#_Toc285059720)

[2.8.2 Операции, вычисления, программы 159](#_Toc285059721)

[2.8.3 Языки программирования. 171](#_Toc285059722)

[2.9 Кодирование знаний 176](#_Toc285059723)

# Информация

Прежде чем начать разговор о *кодировании* информации следует разобраться в том, что же такое *информация*? Далее мы дадим несколько качественных и формальных определений этому понятию, посмотрим на него с различных точек зрения, выявим его свойства и попробуем дать количественную оценку некоторым из них.

## Определения информации

Исторически слово информация (от лат. informatio) пришло в наш язык из латинского и дословно означает «осведомлять». Но время не стоит на месте, и теперь практически в каждой прикладной области используется свое «удобное» определение этому термину.

Например, характерными определениями информации в экономике являются следующие:

* информация это сведения, данные, значения экономических показателей, являющиеся объ­ектами хранения, обработки и передачи и используемые в процессе ана­лиза и выработки экономических решений в управлении;
* информация это один из видов ресурсов, используемых в экономических процессах, получение которого требует затрат времени и других видов ресурсов, в связи с чем эти затраты следует включать в издержки производства и обраще­ния.

В толковом словаре по вычислительной технике В. Иллингуорта, Э.Л. Глейзера и И.К. Пайла приводится определение, удобное для специалистов в области информатики и вычислительной техники, которое мы примем за основу:

Определение 1.1: «Информация – это последовательность *символов*. В свою очередь, *символы* определяются как образы, несущие смысловую нагрузку»■

Примеры определений можно приводить очень долго, но, конечно, «универсальное» для общества определение этому термину следует искать в области закона. Интересен тот факт, что хотя в гражданском кодексе (ст. 128) и указано, что информация является одним из видов гражданских прав, но определения, что такое информации в нём нет. Четкое определение находим в достаточно специфичном Федеральном законе РФ №149 от 27 июля 2006 г «Об информации, информационных технологиях и о защите информации»: «*информация* это сведения (сообщения, данные) независимо от формы их *представления*».

Информации сложно дать единое, максимально общее качественное определение в силу фундаментальности понятия. «Отец кибернетики» Норберт Винер дал, например, такое определение: «Информация есть инфор­мация, а не материя или энергия». Но отделяя понятие информации от материи и энергии, конечно нельзя забывать о тонкой взаимосвязи этих понятий. *Создание, передача, сохранение, копирование, обработка, защита, уничтожение* и прочие действия в отношении информации требуют затрат и материи и энергии.

Одна и та же информация может быть *представлена* в различной материально-энергетической форме, например, в световых, звуковых и радиоволнах, в уровнях электрического тока или напряжения, в напряженностях магнитного поля, в знаках и изображениях на подходящей поверхности и т.д. С помощью весьма ограниченного набора органов чувств (зрение, слух, обоняние, вкус, осязание, чувство равновесия) человек воспринимает бесконечно малую толику из всех возможных форм *представления информации*. И после совершенно незаметного чуда, превращающего воспринятое в *информацию*, законопослушному её обладателю становится ясно, что с точки зрения закона уже совершенно не важно, как эта *информация* была *представлена* до её получения.

## Точки зрения на информацию

Если принять определение, что информация это набор *символов*, которым соответствуют образы, несущие некоторую смысловую нагрузку, то такой набор удобно рассматривать с трех точек зрения.

*С поведенческой точки зрения* важным является то, что любые *действия* (например, создание, получение, передача, обработка и т.д.) над информацией имеют определенные *причины*, а результат этих действий повлечет в свою очередь определенные *следствия*. Например, можно оценивать поведение мыслящего существа или отклики автоматической системы вследствие получения ими определенной информации.

*С математико-лингвистической точки зрения* важна структура набора символов и его смысловое содержание. Обладая *информацией* о структуре и соотнося её с конкретным набором символов, мы можем выделять из этого набора подгруппы символов, несущих совокупную смысловую нагрузку, и таким образом переходить к смысловому содержанию всего набора в целом. Например, с этой точки зрения смотрят на информацию, разбираясь в исходном тексте программы, или анализируя формат файла, содержащего музыкальные или графические данные и т.д.

*С физико-технической точки зрения* взгляд направлен, очевидно, на физические и технические аспекты *представления* информации. То есть важно, в какой материально-энергетической форме информация представлена, какие используются технические способы воздействия на представление информации, каковы количественные оценки информации с точки зрения технических ограничений и т.д. Взгляните с этой точки зрения, например, на процесс записи концерта: как преобразовать информацию, представленную в колебаниях воздуха, в эквивалент на flash-накопителе вашего mp3 плеера?

Авторы в дальнейшем изложении постараются рассматривать информацию со всех указанных точек зрения, но, в силу специфики избранной предметной области, взгляд, конечно, будет несколько «математико-лингвистически-однобоким».

## Свойства информации

Обычно, рассматривая нечто как объект, полезно выделить *внутренние* и *внешние* свойства этого объекта. *Внутренние* свойства органически присущи объекту и характеризуют его вне зависимости от его взаимодействия с другими объектами. *Внешние* же свойства характеризуют объект при его взаимодействии с другими объектами и не имеют смысла в отсутствие такого взаимодействия.

Рассматривая информацию как объект (см. Рис. 1.1), мы выделим два важнейших внутренних её свойства, которым в дальнейшем уделим самое пристальное внимание, – это *количество* (или объем) и *структура* (или внутренняя организация) информации.

О внешних свойствах придется говорить в контексте её отношений с другими объектами, из которых особо мы выделим следующие:

* *отправитель*, он же автор, создатель, источник;
* *получатель,* он же приемник или потребитель;
* *материально-энергетический носитель* информации, без которого, как уже ясно, отправитель не отправит, а получатель не получит информацию;
* *отражаемый объект*, некоторый объект, в отношении которого между *отправителем* и *получателем* идет информационный обмен.



Рис. 1.1 Внутренние и внешние свойства объекта-информации

Раскроем вкратце суть некоторых *внешних* свойств информации. Часть из этих свойств можно определить количественно, то есть *измерить*, выразить числом, а часть можно определить только на качественном уровне.

Самая многочисленная группа *внешних* свойств проявляется в отношении *получателя*. *Полнота* – свойство информации исчерпывающе характеризовать *получателю* *отражаемый объект*. *Релевантность* – свойство соответствовать нуждам (запросам) *потребителя*. *Своевременность* – это *релевантность* в *нужный* момент времени. *Конфиденциальность* – недоступность третьим лицам[[1]](#footnote-1) (не исключает *доступность*). *Доступность* – свойство информации быть доступной законному потребителю (т.е. первым и вторым лицам, что, конечно не исключает *конфиденциальности*). *Целостность* – свойство неизменности относительно некоторого фиксированного значения (например, это свойство дает *получателю* уверенность в том, что он получил информацию в том виде, в котором она была отправлена). *Эргономичность* – удобство избранных свойств (как внешних, так и внутренних) информации для потребителя. *Назначение* – свойство отражающее полезность информации для потребителя-специалиста в определенной области (экономическая, техническая, биологическая, статистическая, генетическая и пр. информация).

В отношении *отправителя* можно выделить следующие свойства информации. *Живучесть* – свойство информации, находясь в *источнике*, сохранять свои *внутренние* свойства с течением времени. *Уникальность* – уверенность в существовании единственного *источника* информации, никогда не осуществлявшего её передачу. *Принадлежность* – свойство информации, позволяющее определить её *источник* илифакт её получения *приемником* (в этом случае это внешнее свойство проявляется в отношении *отправителя* и *получателя* сразу).

Одним из важнейших свойств информации в отношении *отражаемого объекта* является *адекватность*. Понятие *отражаемый объект* следует понимать максимально широко: это может быть все, что угодно: другая информация, процесс, явление, ситуация, предмет и т.д. *Адекватность* – это свойство информации однозначно соответствовать *отражаемому объекту*. *Способ кодирования* – способ, определяющий возможность перехода от *отражаемого объекта* к информации (обратный переход от информации к объекту называется *декодированием*). *Избыточностью* называется превышение *количества* информации, над необходимым минимумом для *адекватного* соответствия *отражаемому объекту*.

В отношении *материально-энергетического носителя* очевидное свойство – *форма представления* информации (собственно сам носитель этой формой и является). *Сигнал* – это изменение (во времени или пространстве) физической величины, несущее *информацию*, т.е. способ позволяющий фиксировать *символ* в материально-энергетическом носителе (напомним, что результатом *кодирования* является набор *символов –* что есть *информация*).

Особое внимание мы уделим способам *кодирования*, которые позволяют минимизировать *количество* информации, соответствующее отражаемому объекту, и гарантировать такие важные для *потребителя* свойства, как *целостность*, *принадлежность и своевременность*.

## Подходы к количественной оценке свойств информации

Как мы уже знаем, единой *точки зрения* на *информацию* нет, нет и единого подхода к оценке свойств информации. Выделяют следующие подходы: *синтаксический*, *семантический*, *прагматический*. *Синтаксический* подход дает оценку *информации*, учитывая только *структурные* особенности *отражаемого объекта*. *Семантический* дает оценку, учитывая смысловую нагрузку, которую несет *отражаемый объект* для *потребителя* его информационного отражения. *Прагматический* же подход дает оценку, учитывая *полезность отражаемого объекта* для *потребителя* в процессе достижения некоторой *цели.*

Многие свойства *информации* (как внутренние, так и внешние) можно оценить *количественно*. На примере такого *внутреннего* свойства информации как её *количество*, покажем, что задача количественной оценки отнюдь не проста.

Если мы примем определение, что информация – это *совокупность* *символов*, то её *количество* оценивается тривиально (см. Определение 1.1). Количество информации – *количество символов* в *совокупности*. Единица измерения – *символ*.

Казалось бы: вот и все! Чего еще думать? Думать действительно нечего, если вы *потребитель* информации. Но вот если вы *создатель* (*автор*), то подумать придется: в отношении *создателя* внутренне свойство «*количество*» не проявляется столь явно в силу того, что самой *информации* попросту пока нет! Есть только *отражаемый объект*, и требуется создать его отражение – *информацию.* Каково будет её *количество* – открытый вопрос.

Забегая вперед, отметим, что мы привыкли измерять информацию в *битах*. *Битом* называют любой *символ* измножества, состоящего двух различных *символов* (обычно ***0*** и ***1***). Смысловая нагрузка *бита*: ответ либо *да* (символ ***1***), либо *нет* (символ ***0***) на вопрос об *отражаемом* объекте. На основе *бита* выстраиваются более крупные единицы информации: *байты*, *килобайты*, *кибибайты*, *мегабайты*, *мебибайты*, и тд… Подробнее об этом далее.

Допустим, вам принесли цифровую стереофотографию, то есть *информацию*, объемом 127 мегабайт. У вас не возникает затруднений оценить, получится ли её сохранить на своем компьютере. Но оглянитесь вокруг. Сможете ли вы оценить объем *информации*, хотя бы с точностью до десятка мегабайт, которая необходима для того, чтобы *отразить* в ней объемную фотографию окружающей вас обстановки?

Итак, трудности представляет оценка того, какое *количество* информации несет в себе тот или иной *отражаемый объект*? Точнее: какое *количество* этой *информации* необходимо и достаточно, чтобы *адекватно* отразить объект для *потребителя*? Поэтому далее мы будем говорить о *количестве информации* именно в контексте *создания* информации, то есть перехода от *отражаемого объекта* к *информации*. Этот переход называется *кодированием*.

Оценку *количества* информации, необходимого для отражения, наиболее четко дает *синтаксический* подход. В действительности же все указанные подходы количественно оценивают внешнее свойство *адекватности* информации. *Синтаксический* подход оценивает *адекватность* отражения структурных свойств объекта. *Семантический* подход оценивает *адекватность* отражения объекта в контексте смысловых единиц, знаний, которыми обладает *потребитель*. *Прагматический* же подход дает оценку *адекватности* информации в процессе достижения *потребителем* определенной *цели*.

### Синтаксический подход

Особое внимание мы уделим *синтаксическому* подходу. В подавляющем большинстве практических случаев *кодирование* осуществляется на его основе. В рамках этого подхода упрощенно рассмотрим оценки американских ученых Ральфа Хартли и Клода Шеннона.

*Отражаемый объект* рассматривается как *источник*, но не *информации*, а *событий*, снимающих *неопределенность* о его *структуре*. В качестве примера приведем задачу о картах.

Пример 1.1 Задача о картах. Имеется колода из восьми карт. По две карты (допустим, туз и двойка) каждой масти. Профессор вытягивает наугад карту и готов честно давать ответы *да*/*нет* на любые задаваемые вопросы. Нужно минимальным количеством вопросов угадать вытащенную карту. ■

Итак. Имеем *отражаемый объект* – это колода карт, из которой вытащили одну карту. *Неопределенность* о *структуре* этой неполной колоды снимается, когда становится ясно, какая карта была вытащена. *Источником событий* служит профессор, а его ответы *да*/*нет* являются *событиями*.

Можно решить задачу ГСН[[2]](#footnote-2) методом: задавать вопросы о конкретной карте. Тогда можно угадать карту и одним вопросом, но вероятность тому весьма мала. В худшем же случае потребуется семь вопросов.

Более «тонкое» решение: каждым вопросом сокращать неопределенность вдвое. Например, ответ на первый вопрос о цвете масти разделит нам исходную колоду на две равные части по четыре карты. Вопрос о конкретной масти оставит нам две карты. И вопрос о достоинстве карты будет последним. См. Рис. 1.2.



Рис. 1.2 Решение задачи о картах

А теперь выполним переход к *информации*. Каждому *событию*-ответу мы поставим в соответствие *символ*: событию *да* – символ *1*, событию *нет* – символ *0*. Двигаясь от вершины полученного дерева (Рис. 1.2) до конкретной карты, формируем соответствующую ей цепочку символов. См. Рис. 1.3. Теперь цепочка из трех символов-бит полностью отражает *структуру* колоды карт. Итак, исходя только из *структурных* свойств колоды, мы выполнили оценку *количества* необходимой информации: три бита.



Рис. 1.3 Переход к информации в задаче о картах

Теперь, чтобы не утруждать себя вопросами, можно попросить профессора так же честно сообщить *информацию* о вытащенной карте. И если он передал вам записанную на бумажке цепочку символов *011* (одно сообщение!), вы без труда догадаетесь, что в колоде не хватает туза треф.

Обобщая пример, вернемся к мере Ральфа Хартли. Пусть у нас имеется конечный набор *символов*. Количество *символов* в наборе – . Количество возможных комбинаций структуры (будем говорить просто: *количество состояний*) отражаемого объекта – . Итак, одним символом можно отразить состояний, двумя , тремя , и т.д., цепочкой длины можно отразить состояний. Конечно, в практических целях нам нужно будет подобрать цепочку такой длины , чтобы . Мера же информации Ральфа Хартли, которая вовсе не обязана быть целым числом, предполагает :

|  |  |
| --- | --- |
| . | (1.1) |

В авторитетных источниках можно встретить такой её вариант:

|  |  |
| --- | --- |
|  | (.) |

где - мера Хартли, – обозначение натурального (по основанию e) логарифма от .

В случае примера с картами: количество состояний , количество символов . Количество информации бита.

Примерно через двадцать лет после работ Ральфа Хартли, его соотечественник Клод Шеннон ввел в 1948 г. более общую меру. При этом мера Хартли не утратила своей ценности и является важным частным случаем оценки по Шеннону. Приведем основные постулаты меры Шеннона.

1. *Информация* есть непрерывная функция от вероятности[[3]](#footnote-3) *события*.
2. *Информация* одиночного -го *события* , ( - множество всех событий), ( – количество различных событий в ), происходящего с вероятностью , имеет положительное значение . Причем
3. Совместная *информация* двух независимых событий с совместной вероятностью , равна сумме их информаций ; .

Итак, *количество информации* зависит от *вероятности* наступления *события*. Сравните свои эмоции от двух событий: «Жучка укусила Иванова» и «Иванов укусил Жучку». Первое событие, хоть собака и друг человека, будничное, а вот второе вызывает улыбку – сенсация! Вероятность первого события весьма велика, вероятность второго близка к нулю. Первое событие несет мало информации, а второе несет большое её количество, отсюда и эмоции.

В то же время, если мы на следующий день услышим, что после Иванова Жучку укусил еще и Петров, то внутренне мы будем готовы к тому, что на третий день Жучку укусит и Сидоров – только ленивый не кусает Жучку… Хотя по правилам теории вероятности, с точки зрения обывателя, который не в курсе отношений Жучки с людьми, вероятность того, что Жучку укусит Иванов также мала, как и вероятность быть покусанной Петровым, и равна . А вероятность, того что Жучка будет покусана обоими, равна *произведению* вероятностей этих событий: – это практически невероятно, так как много меньше ! В то же время никто (разве что Жучка) не падает в обморок от совместных действий Иванова и Петрова, так как количество информации в этом случае есть лишь *сумма* количеств информаций для каждого факта оскорбления действием в отдельности.

Итак, постулатам Клода Шеннона удовлетворяет функция:

|  |  |
| --- | --- |
|  | (1.3) |

где – вероятность *события*, а несет тот же смысл, что и в мере Хартли (см. формулу (1.1)) – количество информационных *символов*.

График зависимости информации от вероятности приведен на Рис. 1.4. При стремлении вероятности события к нулю, количество информации стремится к бесконечности. Событие с вероятностью 1 (происходящее всегда) не несет информации .

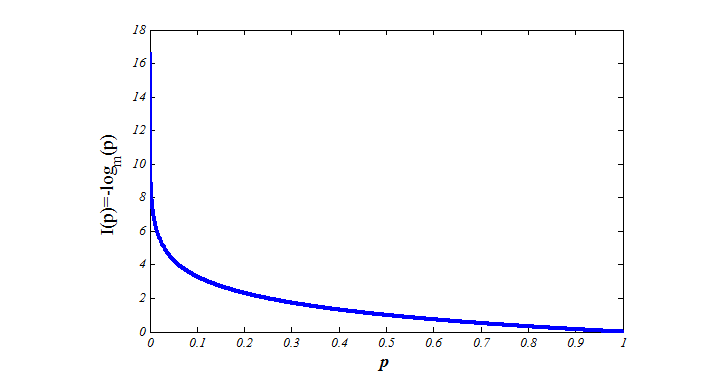


Рис. 1.4 Зависимость количества информации от вероятности

Рассмотрим *источник*, который может выдавать *различных* равновероятных *событий*. Тогда вероятность каждого события будет равна . В качестве примера такого источника, можно взять игральный кубик, при этом вероятность того, что выпадет грань «пять» равняется, очевидно, одной шестой. Если предположить, что *событие* однозначно определяет *состояние* отражаемого объекта, то мера Шеннона (формула (1.3)) превращается в меру Хартли (формула (1.1)):

.

В работе Шеннона вводится важная характеристика *источника событий*: *энтропия. Энтропия –* мера неопределенности источника в целом. Энтропия играет важную роль в оптимальном кодировании, которое подробнее будет обсуждаться в дальнейшем. Проясним смысл понятия на примере источника, для которого известны вероятности поступления событий (источник без памяти в работе Шеннона). Допустим, что мы начали наблюдать за таким источником. Зададимся вопросом: «сколько информации этот источник выдает в среднем?». Если мы наблюдали событий, то событие должно было произойти примерно раз. Если всего источник выдает событий (т.е. ), то за весь период наблюдений он выдал примерно такое количество информации:

В *среднем* же, каждым событием, источник выдавал количество информации равное

|  |  |
| --- | --- |
|  | (1.4) |

Это значение и есть *энтропия* источника. Измеряется она в тех же единицах, что и информация. Максимум энтропии, а значит и неопределенности источника, достигается в том случае, когда все события *равновероятны*. Например, у нас есть источник-монетка: «орёл» выпадает с вероятностью , а «решка», соответственно, с вероятностью . График энтропии для монетки, как зависимость энтропии от вероятности выпадения «орла» представлен на Рис. 1.5. Видно, что максимума энтропия (неопределенность) достигает, когда вероятности выпадения «орла» и «решки» равны 0.5. Это значит что, если вы искренне хотите доверить некоторое решение судьбе, то возьмете монетку с максимальной неопределенностью, а если хотите заработать состояние, то возьмете монетку с нулевой неопределенностью, которая, например, всегда выпадает «орлом», и разорите ближайшее казино.

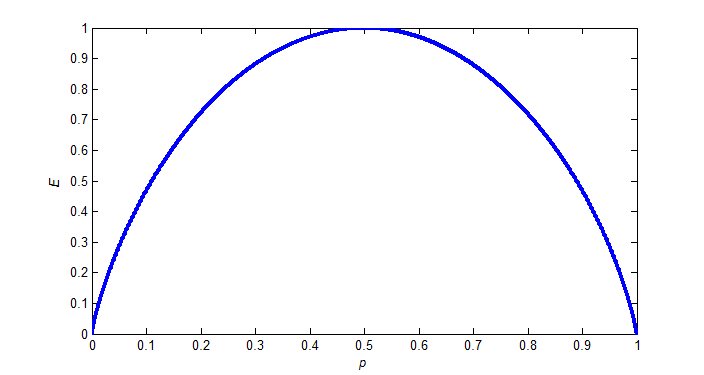
**

Рис. 1.5 Энтропия для источника с двумя состояниями

Почему понятие энтропии столь важно? Создавая *источник информации,* эквивалентный *источнику событий* (то есть, например, в том же порядке выдающий информационные цепочки символов – отражения *событий),* важно помнить, что неопределенность (энтропия) *источника информации* **не может быть меньше** неопределенности *источника событий*. Значение энтропии задает нижний предел неопределенности создаваемого *источника информации*. На практике неопределенность *источника информации* всегда больше неопределенности отражаемого *источника сообщений* и может лишь в той или иной мере к ней приближаться.

Итак, *синтаксический* подход дает нам возможность определять количество информации, необходимое для создания информационного дубликата *отражаемого объекта*. В каких же единицах это количество измеряется? Это зависит от того, сколько различных *символов* используется для составления информационной цепочки. Значение основания логарифма в формуле (1.1) (Хартли) и в формуле (1.3) (Шеннон) определяет единицу измерения. И лишь немногим значениям из бесконечного множества целых чисел история дала имя для соответствующей единицы измерения. О счастливчиках и поговорим.

Когда , единица измерения называется *Бит* (от англ. binary digit; также игра слов: англ. Bit – немного). *Бит* по праву считается элементарной единичкой информации и несет в себе ответ (*да*/*нет*, *истина*/*ложь*) на какой-либо вопрос о предметной области. Обычно истине или ответу «*да*» ставится в соответствие символ 1, а лжи или ответу «*нет*» – 0. На основе бита строятся более крупные единицы, представляющие собой цепочки из нескольких *бит*. Единица *ниббл* (nibble, nybble) равна четырем битам. *Нибблом* можно представить состояний. Россияне привыкли использовать вместо слова «*ниббл*» слово «*тетрада*». По-английски «nibble» означает покусывать, кусать мелкими кусочками. Два *ниббла* составляют *октет*. *Октет* состоит из 8 *бит* и может отразить состояний. В ячейке памяти вычислительной машины хранить такую малую единицу информации как *бит* непрактично, поэтому ячейка памяти хранит, позволяет записывать и считывать как единое целое, сразу несколько *бит*. Эти несколько бит называются б*айтом*. Английское byte – самостоятельное слово и обозначает именно *байт* и ничто другое. Считается, что слово byte произошло от английского bite – кусать. Более строго: *байтом* называют минимальную адресуемую последовательность *битов* в памяти. Например, в IBM-1401 *байт* был равен 6 *битам* так же, как и в Минск-32, а в БЭСМ – 7 битам, в некоторых моделях ЭВМ производства Burroughs Computer Corporation – 9 битам. Так что, строго говоря, сколько *бит* в *байте* зависит от конкретной архитектуры вычислительной системы. В подавляющем большинстве самых распространенных архитектур, *байт* представлен восьмью *битами* и так сложилось, что уже во многих учебниках информатики слово *байт* стало синонимом слову *октет*. Французы, впрочем, предпочитают точность и используют слово «*октет*» вместо «*байт*», когда говорят о *количестве* бит, а не о памяти.

Для формирования более крупных величин из *байтов* (либо из единиц измерения информации, основанных на бите, включая и сам бит) применяют приставки (префиксы), соответствующие сомножителю в виде той или иной степени двойки. По правде сказать, здесь не обошлось без казусов. Не вдаваясь в тонкости, все случилось из-за того, что (1000) очень близко по значению к (1024). И при измерении количества информации названия префиксов для множителей, в основе которых лежит степень 1000, стали неправильно употребляться для обозначения множителей, в основе которых лежит та же степень, но для 1024. Результат такой вольности – программистов обвешивают на 24 грамма на каждый купленный килограмм. А уж сколько обманутых пользователей, верящих в то, что покупают винчестер объемом, например, 200 гигабайт и получающих на самом деле неформатированную емкость на 73741824 байта меньше. Печально, но бизнес есть бизнес: приставка гига – это все-таки , потому как существует интернациональная система единиц измерения (СИ/SI) и это знает каждый школьник. Но школьник знает, а вот знатоки информационных технологий располагают. И породили они путаницу международного масштаба. Еще в 1998 году международное бюро мер и весов (International Bureau of Weights and Measures - BIPM) опубликовало статью, в которой, среди всего прочего, были приведены десятичные префиксы интернациональной системы, и в качестве примера указано, что килобит это 1000 бит, а не 1024! Точка. 19 марта 2005 года был принят стандарт IEEE (Institute of Electrical and Electronics Engineers) за номером 1541, целью которого является введение специальных двоичных префиксов. Повсеместное использование предписано начать не позднее, чем с 2007 года. См. Таблица 1.1.

Таблица 1.1: Сравнительная таблица десятичных (SI) и двоичных (IEEE 1541) префиксов

|  |  |  |  |
| --- | --- | --- | --- |
| Множитель | Префикс СИ/SI | Множитель | Префикс IEEE 1541 |
|  | **kilo**byte (kB) **кило**байт |  | **kibi**byte (KiB) **киби**байт |
|  | **mega**byte (MB) **мега**байт |  | **mebi**byte (MiB) **меби**байт |
|  | **giga**byte (GB) **гига**байт |  | **gibi**byte (GiB) **гиби**байт |
|  | **tera**byte (TB) **тера**байт |  | **tebi**byte (TiB) **теби**байт |
|  | **peta**byte (PB) **пета**байт |  | **pebi**byte (PiB) **пеби**байт |
|  | **exa**byte (EB) **экса**байт |  | **exbi**byte (EiB) **эксби**байт |
|  | **zetta**byte (ZB) **зетта**байт |  | **zebi**byte (ZiB)**зеби**байт |
|  | **yotta**byte (YB) **йотта**байт |  | **yobi**byte (YiB) **йоби**байт |

Надо отметить, что в Российском государственном стандарте ГОСТ 8.417-2002, в приложении А оговаривается, дословно: «…исторически сложилась такая ситуация, что с наименованием «*байт*» *некорректно* использовали (и используют) приставки СИ…». С 2007 года вопрос с подвохом: «сколько байт в килобайте?» неактуален: в *кило*байте 1000 байт, ровно столько же, сколько грамм в *кило*грамме. В *киби*байте 1024 байта.

При , количество информации измеряется в *натах* по той простой причине, что логарифм с основанием называется натуральным. Известно, что основание – это основание оптимальной (с точки зрения компактности представления чисел) позиционной системы счисления (о кодировании чисел см. далее). Это число, увы, иррациональное и использовать его на практике сложно, что не мешает, однако, активно использовать *нат* в теоретических выкладках.

это наиболее близкое к оптимальному целое основание логарифма. Единица измерения количества информации в этом случае называется *трит*. По сравнению с *битом*, *трит* более ёмкая единица и может давать *три* ответа на вопрос: «*да*», «*нет*», «*не знаю*» (*больше*/*меньше*/*равно*, *направо/налево/прямо* и т.д.). Обычно ответу «да» ставится в соответствие символ p (positive) или + (plus), ответу «нет» – n (negative) или - (minus), ответу «*не знаю*» - 0 (zero). В 1956-58 гг в МГУ им. М. В. Ломоносова Николаем Петровичем Брусенцовым с группой единомышленников была создана единственная в мире троичная ЭВМ «Сетунь», получившая название по имени протекавшей рядом речки (*триты* представлялись соответственно нулевым, отрицательным и положительным уровнями напряжения). Была создана память, адресовавшая *трайты* – группы по шесть *трит*. Увы, история положила в основу современной вычислительной техники *биты*, а не *триты*, сделав выбор в пользу большей надежности, вместо оптимальности.

порождает единицу измерения *дит*. Dit от английского **d**ecimal dig**it**. Эта единица известна миру под несколькими именами: dit, он же ban, он же hart или hartley. Человечество активно использует *диты* для кодирования чисел.

Другие значения столь редко использовались, что если у соответствующих им единиц измерения количества информации и были имена, то они растворились во времени.

Мера Хартли (равно как и мера Шеннона) дает способ пересчета одних единиц измерения в другие. Допустим одной единице измерения соответствует основание , а второй – . В этом случае и , следовательно . Таким образом: . Например, одному *диту* соответствует *бита*.

В заключение рассказа о синтаксическом подходе, попытайтесь выполнить оценку необходимого количества информации для отражения следующей ситуации.

Пример 1.2 Задача о биллиардных шарах. Имеется восемь биллиардных шаров с номерами 1-8 соответственно. Все шары одинаковой массы, кроме одного, который тяжелее остальных. Имеются весы фемиды (чашечные). Какое количество взвешиваний вам потребуется, чтобы определить номер тяжелого шара? ■

По аналогии с предыдущим примером (Пример 1.1) вы решили, что 3? Вы ошиблись. Весы фемиды выдают три возможных *события* при взвешивании: «легче», «равновесно», «тяжелее». Результат взвешивания представим *тритом*. Какова длина цепочки *трит*, позволяющая полностью прояснить *структуру* объекта (набора шаров)? Одно взвешивание сокращает пространство поиска *втрое*! Нам нужно разбить набор на три части: два набора по три и один по два шара. Первым взвешиванием сравним на весах два набора из трех шаров. Если наборы равны по весу, то вторым взвешиванием определяем который из двух шаров из набора, не участвовавшего во взвешивании, тяжелее. Если по результатам первого взвешивания один из наборов оказался тяжелее, то вторым взвешиванием сравниваем любые два шара, входящие в этот тяжелый набор… Результат второго взвешивания полностью проясняет ситуацию. Вы совершенно справедливо можете заметить, что можно среди *девяти*, а не среди восьми *шаров* найти более тяжелый! Увы, тогда никого бы не удалось сбить с толку… Переход к информации представлен на Рис. 1.6., где события «тяжелее», «легче», «равновесно» (в отношении левой чашки) отражаются *символами* p,0,n соответственно.



Рис. 1.6 Переход к информации в задаче о биллиардных шарах

Итак, для отражения ситуации с 9 шарами требуется 2 *трита* информации. Это полностью согласуется как с формулой Хартли (), так и с формулой Шеннона, если принять, что любой из девяти шаров с равной вероятностью () может оказаться тяжелее остальных:

[*трит*].

Выбор количества информационных символов , а, следовательно, и единиц измерения информации, зависит от многих факторов. Так как современная вычислительная техника является двоичной (основанной на *битах*), то максимум внимания мы будем уделять случаю .

*Синтаксический* подход к оценке *количества информации* является наиболее теоретически обоснованным среди всех прочих, и активно применяется на практике. Напомним, что он дает оценку *количества информации* в отрыве от таких «человеческих» (а стало быть, весьма субъективных) понятий как «*смысл*» и «*цель*», опираясь только на *структурные* особенности *отражаемого объекта*.

### Семантический подход

Отражаемый объект рассматривается как источник *сообщений*, которые для потребителя этих сообщений несут определенную *смысловую* нагрузку. Поэтому оценка *адекватности информации* зависит в первую очередь от *способности потребителя* принимать (понимать, воспринимать) *сообщения*[[4]](#footnote-4).

Обсуждая *синтаксический* подход, мы приводили пример двух сообщений, несущих разное (с точки зрения *синтаксического* подхода) количество информации: «Жучка укусила Иванова» и «Иванов укусил Жучку» и просили сравнить ощущения. Теперь сравните свои ощущения от следующих двух *сообщений*: «бутявка укусила калушу» и «калуша укусила бутявку»[[5]](#footnote-5). Не доходят сообщения? Ноль *адекватной* информации в обоих случаях? Вероятно, это происходит потому, что в вашем личном тезаурусе нет определения смысла для «бутявки» и «калуши», а также неизвестно в каких отношениях они находятся.

Слову «*тезаурус*» есть много определений. От древнегреческого: «словарь с примерами», до современного лаконичного: «совокупность выражающих смысл единиц языка с описанием отношений между ними». Оценки на основе *тезауруса* получили наибольшее распространение. При этом в большинстве случаев предполагается, что получатель получает именно *информацию* (цепочку *символов*), а не наблюдает *события* (то есть *отражение* уже свершилось). Следовательно, под *сообщением* понимается *информация*, а не *событие*.

Тезаурус пользователя мы обозначим . Если пуст (), то пользователь не понимает (не воспринимает) информацию и для него она – информационный шум. Ежели всеобъемлющ (), то пользователь знает все, и ему нет необходимости понимать (воспринимать) информацию, так как она уже не обогатит его тезаурус.

В большинстве случаев тезаурус пользователя находится между этими крайностями () и в этом случае поступающую *информацию* можно разделить на три части. Допустим, что потребитель получает информационную цепочку определенной длины (). Причем в этой цепочке можно выделить фрагменты, несущие смысловые единицы и отношения которые: уже имеются в тезаурусе пользователя (); отсутствуют в тезаурусе, но поняты (восприняты) и обогатят тезаурус (); не поняты и представляют собой бесполезный информационный шум (). Если длина фрагмента , то поступающая информация идеально согласована с тезаурусом пользователя. При этом важной основной оценкой является *коэффициент содержательности*

Как видно, этот коэффициент представляет собой отношение *количества* новой, осмысленной *информации* к общему её *количеству*.

Для отдельно взятого сообщения, содержащего весьма большое количество смысловых единиц, мы можем построить примерно такой график зависимости новой информации от имеющегося на момент получения тезауруса :

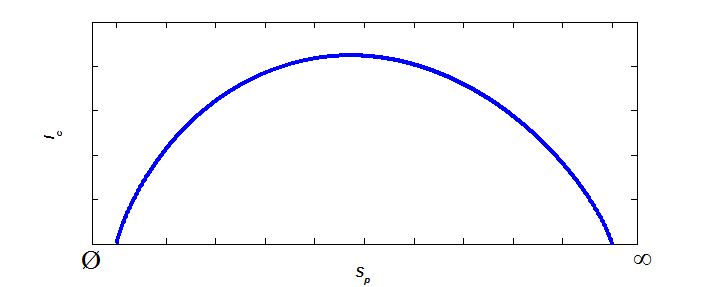


Рис. 1.7 Зависимость новой информации в сообщении от тезауруса получателя

*Семантический* подход учитывает особенности конкретного *потребителя* и дает количественную оценку *адекватности* информации с точки зрения её «осмысленности» для него. Одно и то же *сообщение* может оказаться полезным для компетентного, бессмысленным для некомпетентного и ненужным для всезнающего получателя. Внимание уделяется такой категории, как *знание*, а *информация* в данном случае – это всего лишь *способ* доставки *знаний* до познающего потребителя.

### Прагматический подход

Как уже было сказано, прагматический подход количественно оценивает *адекватность* информации в контексте достижения потребителем какой-либо *цели*. Иными словами, дается количественная оценка *ценности* (*целесообразности*) информации для потребителя. В качестве примера приведем оценку советского ученого Александра Александровича Харкевича (1904-1965):

Здесь – основание логарифма, определяющее единицы измерения (), – вероятность достижения потребителем *цели* до получения информации, – вероятность достижения потребителем *цели* после получения информации. Ценность информации в случае положительна, в случае отрицательна, а в случае равна нулю.

В заключение следует отметить, что результаты оценок *синтаксического* подхода являются наиболее объективными, так как учитывают только *отражаемый объект* и избавлены от влияний со стороны конкретного *потребителя*, что характерно для остальных подходов.

**Вопросы**

1. Дайте несколько определений информации для различных предметных областей. Дайте формальное определение информации по Шеннону.
2. Каковы особенности основных точек зрения на информацию?
3. Назовите основные внутренние и внешние свойства информации. В отношении каких объектов проявляются внешние свойства информации?
4. Раскройте смысловое содержание таких свойств информации, как целостность, конфиденциальность, доступность, актуальность и релевантность.
5. Какие существуют подходы к количественной оценке информации. Какие свойства информации можно определить количественно?
6. Сколько бит, нат, трит и дит информации несет событие, происходящее с вероятностью 0.0012?
7. Совместная информация двух независимых событий равна 7 битам. Вероятность одного из событий 0.125. С какой вероятностью происходит другое событие?
8. Сколько байт в мебибите? Сколько нибблов в килобите?
9. Представьте, что у вас имеется несколько необычный игральный кубик: две его противоположные грани (пусть 1 и 6) являются выпуклыми (дефект производства), остальные четыре грани (2 и 5, 3 и 4) идеально плоские. Падая на выпуклую грань, кубик редко остается на ней, скатываясь на плоские грани. Предположим, что поиграв достаточно долго, мы оценили, что вероятность выпадения грани 1 – 0.08, грани 6 – 0.12, выпадение оставшихся граней - равновероятно. Оцените энтропию кубика в битах и тритах, сравните с энтропией «правильного» кубика.
10. Исключительно в методических целях предлагается с особой осторожностью, не засоряя свой тезаурус, определить коэффициент семантической содержательности следующей информационной цепочки символов: «По данным на 15 лютеня 33 года дракона от начала времен цены на винты упали вдвое, мамки, наоборот подорожали на 15%, фланцы, шканцы и шпиндели стоят тех же бабок. В наступившем растопане[[6]](#footnote-6) мерчендайзеры спешно начали креативить, а народные целители предлагают бады с пиперидиновой структурой, гарантирующие, что бутявки будут раскузявейшие, а калушата – помиковны. <Далее следует совершенно нечитаемый набор из 100 символов>». Пробелы и пунктуация имеют значение. Успехов.
11. В забеге участвуют 6 лошадей. Вы первый раз на ипподроме. К Вам подходит некто и сообщает, что администрация заведения бедствует, и на лошади под номером 1 недавно вспахали поле в ближайшем колхозе, до финиша она пойдет пешком. Дайте в битах прагматическую оценку полученной информации по Харкевичу, причем учтите, что сведения могут быть как правдой, так и ложью, но Вы им все равно поверите…

# Кодирование

Прежде чем продолжить разговор, четче разделим некоторые понятия, которые плотным кольцом окружают обсуждаемую тему кодирования. Под *кодированием* мы будем понимать процесс перехода от *отражаемого объекта* к *набору символов* (*информации*). При этом *отражаемый объект* – это источник значимых для нас *событий*. Ничто не мешает рассматривать *набор символов* (*информацию*) как *отражаемый объект*, когда требуется *закодировать* исходный *набор символов* по-другому (например, уменьшить его размер, сжать).

Другим важным понятием является *сигнал*. *Сигнал* – это изменение (во времени или пространстве) физической величины, несущее *информацию*, т.е. способ, позволяющий фиксировать *символ* в материально-энергетическом носителе. Различают *аналоговые* (непрерывные) и *цифровые* (дискретные) сигналы. Соответственно различают аналоговую и цифровую технику. В чем принципиальная разница между аналоговым и цифровым сигналом?

Представьте, что нужно передать, например, *звук*. Звук это *сигнал* – изменение физической величины «давления воздуха» в определенной точке (пусть в ухе) во времени. Это непрерывное изменение: для любого момента на оси времени можно определить значение давления. Увы, этот *сигнал* для передачи на большие расстояния не годится, требуется *информацию* передаваемую звуком *представить* другим *сигналом*, более удобным для передачи на большие расстояния.

Итак, путь первый, *аналоговый*. Заместить исходный сигнал более подходящим *аналогом*. Например, раскрутить со скоростью 33 оборота в секунду виниловый диск и алмазной иглой, которая благодаря мембране, повторяющей колебания воздуха, прочертит на виниле спиральную дорожку – пространственный аналог звука. Дальше такой сигнал можно передавать самолетом или поездом. Конечно, способов создать аналог масса, например, можно модулировать радиоволну звуковыми колебаниями. См. Рис. 2.1. Конечно, корень *аналог* не подразумевает, что *аналоговый* сигнал обязательно является *непрерывным*, но исторически так сложилось.

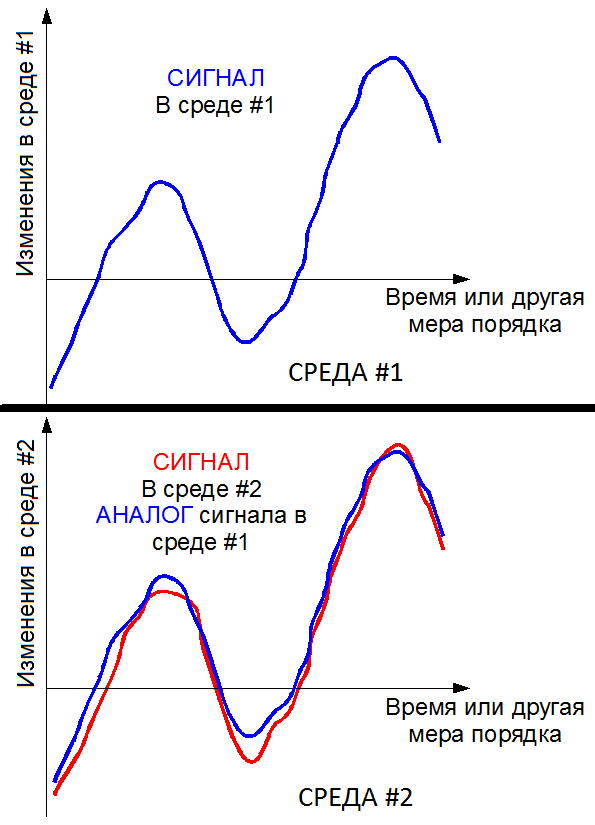


Рис. 2.1 Аналоговый сигнал

*Цифровой* сигнал по определению прерывен, дискретен. Он содержит значения изменяющейся физической величины только в определенных точках времени или пространства. Значение физической величины определяется *числом*. Продолжая пример со *звуком* можно выполнить замеры давления воздуха в определенные моменты времени. Чем меньше промежуток между этими моментами, тем лучше. Получив набор чисел, их можно *закодировать* с помощью конечного набора символов-*цифр.* На Рис. 2.2 числа закодированы в двоичной системе счисления (используются только два символа-цифры «1» и «0»).

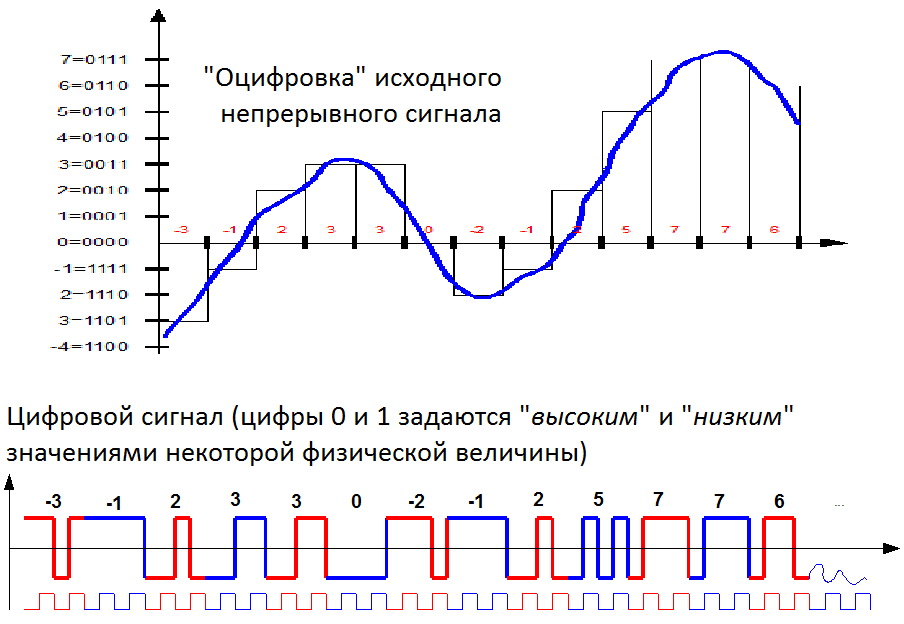


Рис. 2.2 Цифровой сигнал

Естественно, *цифровая* форма представления по определению менее качественна, но обладает одним замечательным достоинством: число можно закодировать на носителе с устойчивыми состояниями (2-мя в современной технике), которые практически невозможно спутать. Поэтому цифровой сигнал намного более устойчив к помехам. Тысячная копия с копии лазерного диска (цифровой сигнал) будет идентичной оригиналу, а уже на сотой производной копии грампластинки вы не услышите ничего, кроме шума. Еще одно преимущество цифрового кодирования информации – возможность использования всей мощи математики, для представления и обработки информации. Вычислительный узел, изначально предназначенный лишь для проведения сложных математических расчетов, в наше время играет ведущую роль в любых манипуляциях с цифровым представлением информации (например, очистка от помех, сжатие и т.д.)

Цифровая техника в настоящее время сильно потеснила аналоговую, но ценители качества, например меломаны, тяготеют к изделиям, в названии которых еще остается вхождения заветного корня «аналог».

И хоть в процессе перехода от одного *сигнала* к другому иногда требуется *кодирование* (сигнал рассматривается как *отражаемый объект*) определения *цифровое* и *аналоговое* к кодированию отношения не имеют.

Напоследок приведем некоторые назначения *кодирования*:

1. принципиальная возможность описания мира с помощью символов конечного алфавита;
2. устранение избыточности, сжатие информации, экономия памяти и снижение нагрузки на каналы передачи информации;
3. обеспечение помехоустойчивости данных;
4. защита важных свойств информации.

**Вопросы:**

1. Каковы основные назначения кодирования?
2. Определите понятия *информация*, *событие*, сигнал, отражаемый объект.
3. В чем различия между аналоговым и цифровым сигналами?

## Формальное определение кодирования

В общем случае имеется множество (алфавит) *событий*:

где – буква алфавита событий, которые требуется закодировать; обозначает отдельное событие; – количество букв алфавита событий .

Алфавит *кодовых символов*:

где – буква кодового алфавита (символ), а – количество символов в алфавите символов .

Требуется задать соответствие (схему, таблицу кодов) между событием и *кодовым словом* :

причем слово , состоящее из букв алфавита (т.е. набор событий):

будет *кодироваться* символами кодового алфавита как

Множество *кодовых слов* , соответствующих называется множеством *элементарных кодов*.

Например, для кодирования *букв* из множества с помощью *слов* алфавита используем следующее соответствие:

Возьмем исходное слово ему однозначно соответствует цепочка . Но вот цепочке помимо исходного, соответствуют слова и . Однозначное *декодирование* невозможно. ■

Пример, когда декодирование возможно:

По-прежнему: , . Используем следующее соответствие:

Возьмем исходное слово ему однозначно соответствует цепочка . Полученной цепочке соответствует только один вариант и *декодирование* возможно. ■

Более строго, выделяют понятие *разделимой* схемы (таблицы кодов). Таблица кодов является *разделимой*, если любое слово , составленное из элементарных кодов единственным образом разлагается на элементарные коды. При этом в таблице кодов не допускается, чтобы одному и тому же элементарному коду соответствовали различные буквы алфавита событий . Разделимая схема допускает *декодирование*.

Важным частным случаем разделимых схем являются *префиксные* схемы. Схема называется префиксной, если ни один элементарный код из множества не является префиксом другого кода из того же множества. *Префиксом*, *началом* или *приставкой* слова называется слово , если . *Постфиксом* или *окончанием* называется, соответственно, слово . Коды, полученные на основе кодирующего дерева, см. пример такого на рисунках Рис. 1.3, Рис. 1.6, очевидно, являются префиксными.

Наиболее простым вариантом кодирования является *равномерное*, когда все элементарные коды одной длины. В этом случае схема заведомо и *разделимая*, и *префиксная*. При этом, если количество кодируемых событий равняется , а количество кодовых букв равняется , и известно, что события равновероятны (вероятность их равна ), то, в соответствии с мерой Хартли (см. формулу (1.1)) потребуется взять кодовые слова длины :

где – *потолок* – наименьшее целое, большее или равное .

Например, в соревновании участвуют 17 спортсменов. Для регистрации пересечения финишной черты каждому спортсмену выдается RFID брелок. В момент пересечения финишной черты спортсменом, брелок передает двоичный код для идентификации спортсмена. Все брелки передают код одинаковой длины. Какое минимально необходимое количество бит в общем случае должен передать брелок?

■

В ряде случаев в процессе кодирования имеются знания о вероятности возникновения тех или иных *событий*. Если это так, то можно использовать методики *оптимального* кодирования для экономии памяти (или снижения нагрузки на каналы передачи информации).

**Вопросы:**

1. Является ли кодирование , , однозначным? Взаимно однозначным? Возможно ли декодирование? Код является префиксным? Разделима ли схема? Приведите примеры.
2. Закодируйте «равномерно» 17 различных сообщений в битах и тритах.

## Оптимальное кодирование

Одним из важнейших внешних свойств информации в отношении *отражаемого объекта* является *адекватность*. Как же установить какое *количество* информации необходимо и достаточно для *адекватного* отражения объекта? Как наилучшим образом *закодировать* отражаемый объект? *Синтаксический подход* (см. параграф 1.4.1) дает наиболее четкий ответ на эти вопросы.

Как уже известно, важной характеристикой *источника событий* (отражаемого объекта) является его *энтропия*. И также известно, что *источник информации*, адекватный *отраженному объекту* будет иметь *энтропию*, по крайней мере, не меньшую, чем отраженный объект. Напомним формулу энтропии[[7]](#footnote-7):

где – вероятность -го *события* , – количество информационных *символов,* используемых для кодирования, – количество событий. В данном случае определяет количество информации , соответствующее событию . Энтропия *источника информации* будет определяться по формуле

|  |  |
| --- | --- |
|  | (2.1) |

где - это количество информации, которое соответствует символу, полученное в результате кодирования. Иными словами это длина кодового слова , соответствующего событию .

Попробуем разобраться на примере. Пусть имеется источник событий , о вероятности появления которых на его выходе известно следующее:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Событие | А | Б | В | Г |
| Вероятность события | 0.5 | 0.3 | 0.1 | 0.1 |

Закодируем источник *равномерной схемой* в алфавите кодовых символов :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Событие | А | Б | В | Г |
| Вероятность события | 0.5 | 0.3 | 0.1 | 0.1 |
| Код события | 00 | 01 | 10 | 11 |

Видно, что эта схема является *префиксной* и *разделимой*. Сравним энтропию источника событий и источника информации. Для источника событий:

Для источника информации:

Видно, что энтропия источника информации больше. Можно ли её уменьшить, приблизить к энтропии источника событий? Очевидно, что если мы будем кодировать символы с большей вероятностью появления кодом с меньшей длиной, то результаты должны получиться лучше. Попробуем следующую схему:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Событие | А | Б | В | Г |
| Вероятность события | 0.5 | 0.3 | 0.1 | 0.1 |
| Код события | 0 | 10 | 110 | 111 |

Так же как и предыдущая эта схема префиксная и разделимая, но неравномерная. Энтропия источника информации теперь

Результаты много лучше! Так, если запустить источник *информации* на выдачу, например, 100 символов то первый вариант кодирования выдаст нам цепочку длины примерно

А второй примерно

Экономия при том же качестве, как говорится, налицо! Лучший ли это результат? Как же закодировать источник действительно оптимальным образом? Рассмотрим два способа (алгоритма) оптимального кодирования информации: алгоритм Хаффмана и алгоритм Фано. Вначале приведем эти алгоритмы для кодирования в алфавите кодовых символов , затем обобщим их на произвольное количество кодовых символов .

***Алгоритм Хаффмана* для**

1. События сортируются по убыванию вероятности.
2. Два события с минимальными вероятностями объединяются в одно составное событие, которое имеет вероятность, равную сумме вероятностей исходных событий. При этом одно из исходных событий помечается кодовым символом 0, а второе – символом 1. Исходные события исключаются из множества событий, вместо них остается одно составное.
3. Шаги 1 и 2 последовательно повторяются до тех пор, пока все события не склеятся в единственное составное событие (корень), вероятность которого, очевидно, равна 1. После этого кодовое слово для исходного события есть цепочка из кодовых символов, которыми помечены все составные события от корня до ■

Закодируем с помощью алгоритма Хаффмана следующий источник событий.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Событие | А | Б | В | Г | Д |
| Вероятность события | 0.5 | 0.125 | 0.125 | 0.125 | 0.125 |

Ход выполнения алгоритма Хаффмана отражен на рисунке Рис. 2.3. Отсортировав исходные события по убыванию вероятности, убеждаемся, что вариантов для склеивания несколько. Произвольно выберем два события с наименьшими вероятностями (энтропия от этого выбора не изменится). Итак, события Г и Д склеиваются в событие ГД с вероятностью 0.25. Г помечается кодовым символом 0, а Д – 1. Теперь множества событий выглядит так: {А,Б,В,ГД}. Наименьшую вероятность имеют события Б и В, которые и склеиваются в событие БВ с вероятностью 0.25. Событие Б помечено символом 0, а В – 1. Множество событий имеет вид {А,БВ,ГД}. Наименьшую вероятность имеют события БВ и ГД, которые и склеиваются в событие БВГД с вероятностью 0.5. Событие БВ отмечено символом 0, событие ГД символом 1. Множество событий теперь имеет вид {А,БВГД}. Отмечая событие А символом 0, а событие БВГД символом 1, приходим к единственному событию в множестве событий {АБВГД} с вероятностью 1.



Рис. 2.3 Кодирование по Хаффману

Далее продвигаясь от корня (события АБВГД) до события исходного множества формируем кодовые слова из символов, которыми помечены промежуточные события на этом пути. Результат:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Событие | А | Б | В | Г | Д |
| Вероятность события | 0.5 | 0.125 | 0.125 | 0.125 | 0.125 |
| Код события | 0 | 100 | 101 | 110 | 111 |

Получен префиксный код. Попробуйте декодировать информацию: 10001101110111. Однозначно?

Попробуйте выполнить алгоритм Хаффмана для примера, на котором мы выполняли оценку энтропии в самом начале параграфа. Сравните также энтропии источника событий и соответствующего ему источника информации для данного примера. Попытайтесь достичь лучших результатов.

***Алгоритм Фано*** **для**

Алгоритм заключается в следующем. На входе алгоритма имеется исходный массив событий, отсортированный в порядке убывания соответствующих им вероятностей (предполагаем, что событий больше, чем , иначе кодирование тривиально – по одному символу на событие). Далее исходный массив разбивается на две части, так, чтобы разница сумм вероятностей событий каждой части была минимальна. Первый кодовый символ элементарного кода для каждого события находится так: для всех событий левой части разбитого массива первый кодовый символ будет 0, а для всех событий правой части – 1. Второй и последующие кодовые символы определяется так: каждая часть разбитого исходного массива, в которой более одного события, становится исходным массивом, а разбиение с нахождением кодового символа для входящих в нее событий выполняется так же как для исходного массива. ■

Такие алгоритмы, как алгоритм Фано называются рекурсивными. Рассмотрим пример. Дано:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Событие | А | Б | В | Г | Д | Е | Ж |
| Вероятность события | 0.135 | 0.24 | 0.25 | 0.125 | 0.0635 | 0.124 | 0.0625 |

Отсортируем события и выполним алгоритм Фано (см. пояснения далее):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | | | |
| B | 0.25 | 0 | 0 |  |  |
| Б | 0.24 | 0 | 1 |  |  |
| А | 0.135 | 1 | 0 | 0 |  |
| Г | 0.125 | 1 | 0 | 1 |  |
| Е | 0.124 | 1 | 1 | 0 |  |
| Д | 0.0635 | 1 | 1 | 1 | 0 |
| Ж | 0.0625 | 1 | 1 | 1 | 1 |

На первом шаге найдем минимальную разницу сумм вероятностей среди вариантов разбиения на части: В:БАГЕДЖ=|0.25-0.75|=0.5; ВБ:АГЕДЖ=|0.49-0.51|=0.02; ВБА:ГЕДЖ=|0.625-0.375|=0.25; и т.д. очевидно, минимум это вариант ВБ:АГЕДЖ. Первый кодовый символ для событий части ВБ – 0, для остальных событий части АГЕДЖ – 1. Имеем две части, для каждой из которых повторяется тот же алгоритм. Продолжим с частью АГЕДЖ, как с более интересной: А:ГЕДЖ=|0.135-0.375|=0.24; АГ:ЕДЖ=|0.26-0.25|=0.01; АГЕ:ДЖ=|0.384-0.126|=0.258; и т.д. минимум – вариант АГ:ЕДЖ. Второй кодовый символ для событий АГ – 0, для событий ЕДЖ – 1. Для каждой части АГ и ЕДЖ повторяется тот же алгоритм. Например, для части АГ имеется единственный вариант разбиения А:Г и третий кодовый символ для А – 0, а для Г - 1. Так ни в одной части количество событий не превосходит единицы и делить уже нечего, то кодовые слова для событий А и Г определены. И так для всех получаемых частей.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Событие | А | Б | В | Г | Д | Е | Ж |
| Вероятность события | 0.135 | 0.24 | 0.25 | 0.125 | 0.0635 | 0.124 | 0.0625 |
| Код события | 100 | 01 | 00 | 101 | 1110 | 110 | 1111 |

Сравните энтропии источника событий и соответствующего ему источника информации самостоятельно.

Мы рассмотрели алгоритмы для двоичного кодирования, когда количество символов . Попытаемся раздвинуть рамки до . Приведем обобщение алгоритма Хаффмана, а алгоритм Фано, любопытный читатель (при желании) обобщит сам.

Итак, приведем очевидное обобщение, алгоритма Хаффмана, которое, к сожалению, не на всех исходных данных даст правильный результат:

***Алгоритм Хаффмана* для**

1. События сортируются по убыванию вероятности.
2. событий с минимальными вероятностями объединяются в одно составное событие, которое имеет вероятность, равную сумме вероятностей исходных событий. При этом первое из исходных событий помечается кодовым символом , а второе – символом и т.д. Исходные события исключаются из множества событий, а вместо них добавляется полученное составное событие.
3. Шаги 1 и 2 последовательно повторяются до тех пор, пока все события не склеятся в единственное составное событие (корень), вероятность которого, очевидно, равна 1. После этого кодовое слово для исходного события есть цепочка из кодовых символов, которыми помечены все составные события от корня до ■

Почему не на всех? Давайте рассмотрим пример. Пусть и

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Событие | А | Б | В | Г |
| Вероятность события | 0.5 | 0.3 | 0.1 | 0.1 |

Итак, по алгоритму получается результат, представленный на рисунке Рис. 2.4 в части а), но очевидно, есть лучший вариант, представленный на том же рисунке в части б).



Рис. 2.4 Кодирование по Хаффману

Результаты кодирования в обоих случаях:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Событие | А | Б | В | Г |
| Вероятность события | 0.5 | 0.3 | 0.1 | 0.1 |
| Код события рис 2.4.а | 0 | pn | p0 | pp |
| Код события рис 2.4.б | n | 0 | pn | p0 |

Энтропия источников информации в случае а) равна 1.5 трит, в случае б) равна 1.2 трит, энтропия же источника событий равна примерно 1.06342. Не доказывая, примем за справедливое без доказательства следующее утверждение: в корневое событие на последнем шаге алгоритма Хаффмана должны склеиваться не менее событий. При этом мы вправе вводить произвольное число «фиктивных» событий (), вероятность возникновения которых равна *нулю*! Сколько же их требуется? При склеивании событий замещаются одним составным, то есть количество событий уменьшается на . Стало быть, после склеиваний на последнем шаге должно остаться событий:

|  |  |
| --- | --- |
|  | (2.2) |

где - количество исходных событий, – количество искусственно вводимых «фиктивных» событий с нулевой вероятностью возникновения, – количество символов кодового алфавита, – количество склеиваний, предшествующих заключительному.

Для приведенного выше примера, исходя из , мы получаем . И на рисунке Рис. 2.4.б приведен корректный результат кодирования по Хаффману.

Итак, прежде чем выполнить обобщенный алгоритм Хаффмана для нужно исходя из соотношения (2.2) ввести «фиктивных» событий. Важно также отметить, что при соотношение (2.2) выполняется всегда при , поэтому никаких «фиктивных» событий при этом не вводится.

**Вопросы:**

1. Закодируйте оптимальным образом источник сообщений, если известны вероятности появления событий на его выходе. Оцените энтропию источника сообщений и соответствующего ему источника информации. Дайте оценку экономии относительно равномерного кодирования. Также закодируйте требуемую последовательность событий. Закодируйте информацию в битах или тритах, используя алгоритм Хаффмана и алгоритм Фано (уточните у преподавателя).

Вариант 1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Событие | A | B | C | D | E | F | G | H |
| Вероятность события | 0.130 | 0.130 | 0.126 | 0.126 | 0.248 | 0.120 | 0.060 | 0.060 |

Последовательность событий: «GHADEFCB».

Вариант 2.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Событие | A | B | C | D | E | F | G | H |
| Вероятность события | 0.333 | 0.111 | 0.111 | 0.111 | 0.111 | 0.111 | 0.038 | 0.074 |

Последовательность событий: «FAGABACABDBCEDEFH».

Вариант 3.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Событие | A | B | C | D | E | F | G |
| Вероятность события | 0.25 | 0.125 | 0.125 | 0.125 | 0.125 | 0.125 | 0.125 |

Последовательность событий: «DEAFABACGA».

Вариант 4.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Событие | A | B | C | D | E | F |
| Вероятность события | 0.25 | 0.25 | 0.125 | 0.125 | 0.125 | 0.125 |

Последовательность событий: «BDEBABAABCBF».

Вариант 5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Событие | A | B | C | D | E |
| Вероятность события | 0.25 | 0.25 | 0.25 | 0.125 | 0.125 |

Последовательность событий: «BCACDABE».

1. Дана последовательность событий: ABCBCDEEDECDDEEAEBCBFCDABDEBECE. Требуется передать её по цифровому каналу связи (цифровой сигнал), так чтобы нагрузка на него была минимальной. Схема кодирования приемнику не важна. Рассмотрите следующие варианты: канал передает информацию в битах, в тритах и в алфавите . Выбор алгоритма кодирования произволен.
2. От двоичного источника информации была получена следующая последовательность символов: «0110100010010101010110111001111011100101110100100110001100». Об источнике информации было известно, что он отражал источник сообщений с приведенными ниже характеристиками (использовался алгоритм Хаффмана и склеиваемое сообщение с большей вероятностью отмечалось символом 0). Восстановите сообщение, составленное из событий.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Событие | А | З | Л | Н | О | П | Р | У |
| Вероятность события | 0.6 | 0.0558 | 0.0442 | 0.0438 | 0.0562 | 0.0538 | 0.092 | 0.0542 |

## Кодирование с целью сжатия

Мы уже рассматривали способы оптимального кодирования информации. Но не всегда информация отражает объект оптимальным образом (иногда в этом нет необходимости) и количество информации превышает необходимый минимум. В этом случае говорят об *избыточности* информации. Но очень часто требуется уменьшить количество информации, сжать её в целях экономии ресурсов памяти или снижения нагрузки на каналы передачи. Методы оптимального кодирования активно используются при *сжатии* информации, так, например, многие алгоритмы сжатия изображений используют алгоритм Хаффмана.

Итак, чётче разделим понятия *оптимальное кодирование* и *кодирование с целью сжатия*. *Оптимальное кодирование* ставит себе в задачу сопоставить *отражаемому объекту* минимальное *количество* *адекватной* ему информации. *Кодирование с целью сжатия* ставит себе в задачу уменьшить *количество информации,* не теряя (или оставаясь в допустимых рамках) при этом свойство *адекватности*. Кодирование с целью сжатия будем далее называть просто *сжатие*. В случае *сжатия,* *отражаемым объектом* выступает *информация*, причем события в этом случае представляют собой *слова* в алфавите . То есть *информация* *перекодируется* в том же алфавите .

Выделяют два больших класса алгоритмов сжатия информации: сжатие *с потерями* и *без потерь*. «Теряется», конечно, *адекватность*. При сжатии без потерь из сжатой информации можно восстановить исходную информацию в точности такую же, как до сжатия. При сжатии с потерями восстановленная информация будет отличаться от исходной. Ярким примером сжатия с потерями является сжатие изображений: используя определенные особенности *восприятия* цвета человеком, такие алгоритмы отбрасывают «лишнюю» информацию. Потеря *адекватности* *отражаемому объекту* в этом случае значительная, но для *человека-потребителя* эти потери адекватности незаметны.

Часто алгоритмы сжатия весьма специфичны и учитывают особенности отражаемого объекта (как в случае изображений). Важнейшим отражаемым объектом в жизни человека является речь, текст (см. параграф 2.5 кодирование текста). На основе, например, русского алфавита можно построить бесконечное количество слов, но в реальной жизни словарный запас редко превышает сотню тысяч слов. На практике для универсального представления текста байтами кодируются буквы, цифры, знаки препинания, пробелы и т.д., но если мы знаем, что кодируется именно *осмысленный* текст (содержащий осмысленные слова), то можно сильно сэкономить, кодируя в качестве сообщений не буквы, а слова. Такие методы сжатия называются *словарными*.

Впрочем, *словарные* методы могут использоваться не только для кодирования текста, но для произвольных информационных цепочек. Причем *словарь* может строиться динамически и совершенно не учитывать смысловой нагрузки слов в словаре.

В этом разделе мы рассмотрим один из методов словарного сжатия: алгоритм Лемпела-Зива. В разделах, посвященных кодированию изображений и звука (например, в параграфе 2.7.4.1) вы можете ознакомиться с другими, более специфичными алгоритмами сжатия.

В основе алгоритма Лемпела-Зива лежит идея *адаптивного сжатия*. За один проход по тексту одновременно строится и словарь и код, причем словарь не хранится, так как при декодировании он динамически восстанавливается. Мы рассмотрим кодирование и декодирование на примере, при этом допустим, что для кодирования букв (цифр, знаков препинания и т.д.) текста используется байт. Для хранения индекса слова в словаре также будет использоваться байт.

Алгоритм кодирования следующий:

1. В словарь нулевым элементом помещается *пустое* слово . *Пустое* слово не содержит букв и для любого слова справедливо .
2. От исходного текста отделяется слово , где – максимально длинное слово из словаря, - расширяющая *буква*. Т.е. .
3. В словарь добавляется новое слово . К коду добавляется пара , где - индекс слова в словаре. От исходного текста отделяется слово : .
4. Пункты 2-3 последовательно повторяются до тех пор, пока в тексте остается хоть одна буква. ■

В результате получается код , представляющий собой упорядоченное множество пар <индекс,буква>: .

Алгоритм декодирования следующий:

1. В словарь нулевым элементом помещается *пустое* слово . Текст не содержит букв: .
2. От исходного кода отделяется пара , в словарь добавляется слово , где – -е слово из словаря (словарь восстанавливается так же, как и формируется!). Текст .
3. Пункт 2 последовательно повторяется до тех пор, пока в коде остается хоть одна пара. ■

Прежде, чем рассмотреть пример, нужно отметить, что данный алгоритм хорошо сжимает тексты большого объема, в которых так или иначе будут присутствовать одинаковые и достаточно длинные слова. В примере такие вхождения созданы искусственно. Для текстов малого объема, скорее всего, длина кода сжатого текста будет превышать длину кода для исходного текста.

Пример. Сжать информацию: «АБАКАНКАНКАНКИАНКИН». Процесс кодирования представлен ниже:

|  |  |  |  |
| --- | --- | --- | --- |
| Шаг | Текст  (**жирным** выделено ) | Словарь  (индекс→слово) | Код  (пара <индекс, буква>) |
|  |  | 0→ε |  |
| 1 | **εА**БАКАНКАНКАНКИАНКИН | 1→A | <0,А> |
| 2 | **εБ**АКАНКАНКАНКИАНКИН | 2→Б | <0,Б> |
| 3 | **АК**АНКАНКАНКИАНКИН | 3→АК | <1,К> |
| 4 | **АН**КАНКАНКИАНКИН | 4→АН | <1,Н> |
| 5 | **εК**АНКАНКИАНКИН | 5→К | <0,К> |
| 6 | **АНК**АНКИАНКИН | 6→АНК | <4,К> |
| 7 | **АНКИ**АНКИН | 7→АНКИ | <6,И> |
| 8 | **АНКИН** | 8→АНКИН | <7,Н> |

Итак, полученный код (сжатый текст) занимает 8\*2=16 байт информации (один байт на индекс, второй на букву), а код исходного текста занимает 19 байт (один байт на букву). Декодирование (восстановление текста из кода) представлено ниже:

|  |  |  |  |
| --- | --- | --- | --- |
| Шаг | Код  (пара <индекс, буква>) | Словарь  (индекс→слово) | Текст  (**жирным** выделено ) |
|  |  | 0→ε |  |
| 1 | <0,А> | 1→A | **εА** |
| 2 | <0,Б> | 2→Б | А**εБ** |
| 3 | <1,К> | 3→АК | АБ**АК** |
| 4 | <1,Н> | 4→АН | АБАК**АН** |
| 5 | <0,К> | 5→К | АБАКАН**εК** |
| 6 | <4,К> | 6→АНК | АБАКАНК**АНК** |
| 7 | <6,И> | 7→АНКИ | АБАКАНКАНК**АНКИ** |
| 8 | <7,Н> | 8→АНКИН | АБАКАНКАНКАНКИ**АНКИН** |

Качество сжатия можно улучшить, например, за счет начальной инициализации словаря (например, инициализировать его буквами алфавита).

**Вопросы:**

1. Проанализируйте алгоритм Лемпела-Зива. Например, дайте ответ на вопрос: когда при добавлении пары на этапе сжатия происходит «сжатие» а когда, наоборот, «расширение»? Приведите примеры, когда «сжатый» текст больше исходного.
2. Выполните сжатие текста «ABABAABAABBAAAABBABAA» с помощью алгоритма Лемпела-Зива.
3. Выполните сжатие текста «AAAAAAAAAAAAAAAAAAAAA» с помощью алгоритма Лемпела-Зива.
4. Реализуйте алгоритм и поэкспериментируйте на больших осмысленных текстах с его улучшениями за счет инициализации словаря.

## Кодирование чисел

*Числа* играют важнейшую роль в жизни человека. Числа задают порядок, дают возможность измерять и сравнивать, определять математические законы, которые отражают законы физического мира и т.д. И только потому, что данный текст посвящен кодированию, числам отведено не первое место. Итак, *отражаемый объект* – *число*.

Числа, как и способы их записи (читай кодирования), пришли к нам из глубокой древности[[8]](#footnote-8). Принципов кодирования чисел и сейчас достаточно много, поэтому мы рассмотрим всего один, наиболее популярный из этих принципов, который читателю знаком со школьной скамьи – *позиционный*.

### Позиционные системы счисления

Позиционная система счисления является способом представления (в частности записи) *чисел* с помощью символов из конечного алфавита – *цифр*. Система счисления – это способ *кодирования* чисел.

*Целое положительное* число в позиционной системе счисления с основанием представляется (записывается) кодовым словом из алфавита цифр следующим образом:

В такой записи на месте находится один из отличимых друг от друга символов кодового алфавита – *цифра*. Каждому такому символу-цифре соответствует *число* из диапазона . Мы будем обозначать *цифрой* и символ, и соответствующее этому символу *число*, понимая из контекста, о чем именно идет речь. Итак, – это -я *цифра* числа . Номер позиции цифры в числе называется *разрядом*.

Число формируется на основе своего кодового слова так:

Впрочем, рано или поздно (в зависимости от величины числа , начиная с некоторого для всех все будут равны нулю:

То есть бесконечный () ряд нулей слева в записи числа можно (лишь для экономии времени записи) опустить[[9]](#footnote-9):

Система счисления с основанием называется -ичной. Одна и та же цифра , записанная в разных позициях, например в -й и -й: , – будет вносить в общую сумму разной величины слагаемые: и соответственно. От того, в какой *позиции* (разряде) стоит цифра в такой записи, зависит всё. Поэтому система счисления и названа *позиционной*. Значение называется *весом* -го разряда.

Несколько примеров.

Кодовому слову (мы привыкли писать просто 78642, так как с детства привыкли кодировать числа десятичной системе счисления) соответствует число ■

Кодовому слову соответствует число ■

Кодовому слову соответствует число ■

В ручной записи *числа* каждой *цифре* соответствует *символ* определенного начертания. Поэтому, если оговорено, что, например и цифра и , то слову соответствует число 106. ■

В вычислительных системах каждой цифре будет соответствовать определенное устойчивое состояние физической среды. Например, в ЭВМ, производящей вычисления в двоичной системе счисления, цифрам 0 и 1 соответствуют низкий и высокий (или наоборот) уровни электрического сигнала.

Дробная часть положительного вещественного (действительного) числа, изменяющаяся, как известно в пределах , будет представлена кодовым словом в позиционной системе счисления с основанием так:

Число при этом на основе своего представления формируется так:

Для записи иррациональных чисел, например таких, как число понадобится действительно бесконечное количество цифр для представления дробной части в системе счисления с *любым* целым основанием. Но, как видно из примера записи , на практике ничего другого не остается как отбросить начиная с некоторого все цифры справа:

Записи дробной части соответствует число ■

Объединяя вышесказанное относительно целой и дробной части можно сказать, что положительное вещественное число X можно представить так:

Используя нотацию Айверсона:

добавим знак:

Записи отрицательного числа будет предшествовать знак минус:

Число – понятие универсальное. Одно и то же число совершенно по-разному будет закодировано в системах счисления с разными основаниями. Кодовое дерево, поясняющее принцип кодирования целых чисел в алфавите (двоичная система счисления) приведено на рисунке Рис. 2.5. Кодовое слово получается при движении от корня до конкретного числа. Очевидно, что схема получается не префиксная и не разделимая, что и не требуется, так как кодовое слово, соответствующее числу, выделяется в тексте явно.



Рис. 2.5 Кодирование целых чисел в двоичной системе счисления

Тем не менее, вычисления (преобразования чисел, например, сложение, вычитание и пр.) реализуются именно как преобразования символов кодовых слов соответствующих чисел. И при этом важно то, в какой системе счисления они представлены. Люди привыкли использовать десятичную систему счисления, и вряд ли у нас с вами получится так же быстро сложить числа 10 и 20, представленные в двоичной системе счисления. Поэтому здесь и далее по тексту, если для числа явно не оговорено, как оно представлено, то используется десятичное представление.

### Перевод чисел из одной системы счисления в другую

Строго говоря, раздел следовало назвать: «перевод кодов чисел из позиционной системы счисления с основанием в позиционную систему счисления с основанием ». Пусть необходимо представить число в -ичной системе счисления. При этом само число записано в -ичной системе, а вычислитель способен выполнять операции в -ичной системе счисления. Например, люди привыкли считать в десятичной системе счисления и, если число записано не в десятичной системе, то ничего другого не остается, как это число в ней представить.

Знак у переводимого в -ичную систему счисления числа останется прежним, поэтому рассмотрим перевод положительного числа, представленного в системе счисления с основанием :

Каждой цифре , а также основанию ставятся в соответствие числа в системе счисления с основанием (это не представляет сложности). Далее вычислитель по формуле

проводит расчет и получает число в -ичной системе счисления.

Пример. Дано число в шестнадцатеричной системе. Необходимо его представить в системе счисления доступной для вычислителя.

Данный текст был написан вычислителем, которому удобно считать в десятичной системе. В шестнадцатеричной системе счисления цифрам от нуля до девяти соответствуют числа десятичной системы, а цифрам от до (в алфавитном порядке), соответствуют числа от 10 до 15. ■

Итак, число представлено в системе, в которой умеет считать вычислитель, и его нужно перевести в -ичную. Допустим, что число уже представлено в -ичной системе:

Очевидно, что, если разделить его целую часть на основание , то в остатке от деления получается значение нулевого разряда, которое однозначно переводится в цифру -ичной системы. Если поделить частное на , то найдем значение первого разряда, как остаток от деления, и так далее:

Итак, каждый следующий разряд числа мы получаем как остаток от частного, полученного на предыдущем шаге.

Теперь представим, что дробная часть числа также представлена в -ичной системе. Умножим дробную часть на :

Целая часть полученного числа представляет собой значение разряда . Оставшуюся дробную часть умножим на , находя таким образом и так далее:

Дано число в десятичной системе. Необходимо его представить в системе счисления с основанием 3.

Переводим целую часть :

Переводим дробную часть (получается периодическая дробь):

Результат перевода: .■

Напоследок остается заметить, что иногда цифре -ичной системы не всегда соответствуют числа из диапазона. Вполне допустимо сдвинуть диапазон влево, в сторону отрицательных чисел: . При этом каждый разряд становится «знаковым» (то есть может быть отрицательным), и знак перед *числом* уже не ставится. Перевод чисел в этом случае станет чуть сложнее, но вдумчивый читатель разберется сам. Далее рассмотрим несколько популярных позиционных систем счисления.

### Двоичная система счисления

Система счисления с основанием Цифры всего две: 0 и 1. Перевод в двоичную систему счисления или из неё не имеет никаких особенностей и осуществляется так же, как и для любой позиционной системы.

Основные арифметические операции над числами реализуются через преобразования их символьных представлений (кодов). Мы рассмотрим преобразования двоичных кодов чисел, соответствующие сложению: после преобразования кодов двух чисел получается код числа, равного их сумме.

Рассмотрим, как выполняется преобразования кодов, соответствующие сложению чисел. Таблица сложения двух цифр () -го разряда представлена далее:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | + |  | = |  |
| 0 |  | 0 |  | 00 |
| 0 |  | 1 |  | 01 |
| 1 |  | 0 |  | 01 |
| 1 |  | 1 |  | 10 |

В результате сложения цифр получается двоичных код числа (проверьте, выполнив перевод двоичных кодов), состоящий из двух цифр , причем представляет собой перенос в следующий разряд. Так как, очевидно, перенос должен складываться с результатом сложения следующего разряда , то удобнее воспользоваться следующей таблицей, учитывающей перенос:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | + |  |  |  | = |  |
| 0 |  | 0 |  | 0 |  | 00 |
| 0 |  | 1 |  | 0 |  | 01 |
| 1 |  | 0 |  | 0 |  | 01 |
| 1 |  | 1 |  | 0 |  | 10 |
| 0 |  | 0 |  | 1 |  | 01 |
| 0 |  | 1 |  | 1 |  | 10 |
| 1 |  | 0 |  | 1 |  | 10 |
| 1 |  | 1 |  | 1 |  | 11 |

Чтобы сложить два двоичных числа и нужно выровнять их по разрядам (чтобы цифры слагаемых одного и того же разряда находились друг под другом). Выполнять сложение начинают с младших разрядов (справа-налево). Результат в текущем разряде получается как сумма цифр слагаемых и переноса из предыдущего разряда.

Сложить числа, которым соответствуют двоичные коды: и .

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Разряд | 3 | 2 | 1 | 0 | . | -1 | -2 | -3 | -4 | -5 | -6 |
| Перенос | 1 | 1 | 1 | 1 | . | 1 | 0 | 1 | 0 | 0 |  |
| Разряд слагаемого : |  | 1 | 0 | 1 | . | 1 | 1 | 0 | 1 |  |  |
| Разряд слагаемого : |  |  | 1 | 1 | . | 0 | 1 | 0 | 1 | 1 | 1 |
| Результат | 1 | 0 | 0 | 1 | . | 0 | 0 | 1 | 0 | 1 | 1 |

Сложение и вычитание – базовые операции, на которых основаны операции умножения и деления. При вычитании чисел используется приемы, позволяющие заменить вычитание сложением. Эти приемы основаны на потере переноса при сложении чисел в конечной разрядной сетке. Рассмотрим, как можно перейти от вычитания к сложению, используя дополнительный и обратный коды.

Допустим, что для представления двоичного числа используется разрядов. В такой разрядной сетке можно представить целые положительные числа от нуля до . Например, если к числу прибавить единицу, то возникнет перенос в -й разряд, этот перенос будет потерян, и результом будет ноль. Конечно в -разрядной сетке можно представить не только целые числа, но и вещественные. Для этого достаточно жестко «зафиксировать» точку (разделитель целой и дробной части) между определенными разрядами. Такой формат представления вещественных чисел называется форматом с фиксированной точкой(запятой)[[10]](#footnote-10).

Известно что. Как же тогда представить отрицательное число ?

В *дополнительном* коде это делается так: -разрядная сетка дополняется двумя «*знаковыми*» разрядами. Получается -разрядная сетка . Как ранее уже отмечалось в такой сетке соответствует нулю. Тогда отрицательное числоможно представить как .

где – инверсия (дополнение до 1) разрядов числа .

называется дополнительным кодом[[11]](#footnote-11) числа . При этом в «знаковых» разрядах будет код 11. Дополнительный код положительного числа есть просто его представление в -разрядной сетке и в «знаковых» разрядах всегда 00. Модули (положительные части) чисел при этом должны быть представимы в -разрядной сетке. Если результат сложения дополнительных кодов числел будет непредставим в -разрядной сетке, то в «знаковых» разрядах будут комбинации, отличные от 00 и 11. Дополнительный код числа будем обозначать .

Причем, исходя из справедливо и

Результатом «сложения» двух двоичных дополнительных кодов чисел, будет дополнительный код результата сложения.

Используя дополнительный код, сложить числа 235 и -156 в двоичной системе счисления. Разрядная сетка .

Используя дополнительный код, сложить числа -235 и 156 в двоичной системе счисления. Разрядная сетка .

Приведем напоследок пример сложения, дающего ошибочный результат из-за переполнения разрядной сетки (когда значащие цифры результата не помещается в отведенное количество разрядов).

Используя дополнительный код, сложить числа -235 и -156 в двоичной системе счисления. Разрядная сетка .

Далее рассмотрим еще один способ перехода от вычитания к сложению – обратный код. Аналогично, как и в дополнительном коде:

– обратный код числа . Обратный код отрицательного числа это инверсия всех разрядов его модуля. Обратный код будем обозначать .

Видно, что справедливо также:

Обратный код прост в получении[[12]](#footnote-12), но в ряде случаев результат сложения обратных кодов чисел будет получен не в обратном коде. Могут возникать следующие случаи, требующие поправок (A и B – положительные числа):

* : . В этом случае результат представлен в обратном коде верно. Переноса из «знаковых» разрядов нет.
* : . В этом случае возможны два варианта.
  + , результат верно представлен в обратном коде . Признаком тому может служить отсутствие переноса из «знаковых» разрядов.
  + , результат неверен (должен быть ). Признаком тому может служить ненулевой перенос из «знаковых» разрядов. Чтобы получить верный результат в обратном коде к полученному числу нужно прибавить единицу.
* : . Результат получен неверно (должен быть (). Признаком тому может служить ненулевой перенос из «знаковых» разрядов. Чтобы получить верный результат в обратном коде к полученному числу нужно прибавить единицу.

Итак, к полученному в результате сложения обратных кодов коду нужно «прибавить» перенос из «знаковых» разрядов. Так же как и в дополнительном коде, признаком ошибки переполнения служит комбинация, отличная от 00 и 00 в «знаковых» разрядах. Приведем несколько примеров.

Используя обратный код, сложить числа 235 и -156 в двоичной системе счисления. Разрядная сетка .

Используя обратный код, сложить числа -235 и 156 в двоичной системе счисления. Разрядная сетка .

Приведем напоследок пример сложения, дающего ошибочный результат (переполнение разрядной сетки). Используя обратный код, сложить числа -235 и -156 в двоичной системе счисления. Разрядная сетка.

Мы рассмотрели сложение, которым после некоторых манипуляций над *кодом* числа заменяется операция вычитания. На основе операций сложения и вычитания строятся операции умножения и деления. Дабы не перегружать тему *кодирования* математикой, более сложные операции и функции не рассматриваются.

Особенностью двоичной системы является то, что цифры можно трактовать как биты информации (истина, ложь). Преобразования двоичных чисел тесно связаны с логическими операциями. Это, а также то, что двоичная техника надежна (нужно различать лишь два устойчивых состояния среды) и обусловило то, что двоичная система счисления используется в подавляющем большинстве вычислительных устройств.

### Восьмеричная и шестнадцатеричная системы счисления.

Эти системы с основанием, кратным степени двойки. Это дает весьма полезные на практике взаимосвязи этих систем с двоичной.

Рассмотрим преставление числа в двоичной системе, разбив его, начиная от точки, на группы по три цифры:

Вынесем за скобки:

Получили запись числа:

где

Так как , то . То есть трехразрядным двоичным числам вполне можно сопоставить цифры восьмеричной системы: .

Для представления восьмеричных чисел в некоторых языках программирования используются следующие соглашения:

* С++, java, C#, и т.д.: если справа перед числом записан ноль, то число в восьмеричной системе. 015720 - восьмеричное число (равное 7120). 0189 - ошибка: недопустимы цифры 8 и 9. Без ведущего нуля число считается десятичным;
* В ассемблере после цифр восьмеричного числа пишется латинская буква «o» (octal). 15720o. Ну и 189o – надо ли говорить…

Совершенно аналогично происходит перевод в шестнадцатеричную систему. При этом исходное двоичное число разбивается от точки на группы по четыре двоичные цифры. Числа от нуля до десяти обозначаются теми же символами, что и в десятичной системе, а оставшиеся числа от десяти до пятнадцати обозначаются латинскими буквами: A, B, C, D, E, F. Соответствие шестнадцатеричных цифр четырехразрядным двоичным числам следующее: .

Для представления шестнадцатеричных чисел в некоторых языках программирования используются следующие соглашения.

* С++, java, C#, и т.д.: если слева от цифр числа есть префикс «0x», то число в шестнадцатеричной системе. 0xAF - шестнадцатеричное число (равное 175). 0x1h - ошибка: недопустима цифра h.
* В некоторых ассемблерах после цифр шестнадцатеричного числа пишется латинская буква «h» (hexadecimal): afh, AFh, AFH.

Видно, что перевод в двоичную систему из шестнадцатеричной или восьмеричной и обратно, тривиален, и потому эти системы счисления (восьмеричная и шестнадцатеричная) активно применяются в языках программирования для повышения компактности записи программ. Компактность же значительно уменьшает количество ошибок, допускаемых человеком. Например, в двоичном коде числа гораздо проще допустить опечатку, чем в восьмеричном или шестнадцатеричном .

Несколько примеров для практики.

Дано двоичное число . Перевести его в системы счисления с основанием 8 и 16. . ■

Дано восьмеричное число . Перевести его в систему счисления c основанием 16. Переводим в двоичную систему:

.■

Дано десятичное число . Перевести его в систему счисления c основанием 2. Переводим в шестнадцатеричную систему (что быстрее, очевидно) целую часть:

Дробную часть:

В двоичной системе:.■

Данные системы счисления облегчают работу с двоичной системой для человека и представлены практически во всех языках, подходящих для системного программирования.

### Форматы чисел в вычислительных машинах

В данном параграфе мы рассмотрим несколько форматов представления чисел в вычислительных машинах, которые в частности используются в популярных процессорах фирмы Intel[[13]](#footnote-13). Форматы представления чисел в процессорах определяются стандартами IEEE (Institute of Electrical and Electronics Engineers – института инженеров электротехники и электроники).

Форматы (типы) представлений чисел представлены в следующей таблице.

Таблица 2.1 - Представление чисел в ЭВМ

|  |  |  |
| --- | --- | --- |
| Число | К-во байт(бит) | Пределы (в 10 СС) |
| Байт (целое) | 1(8) |  |
| Короткое слово (целое) | 2(16) |  |
| Слово (целое) | 4(32) |  |
| Длинное слово (целое) | 8(64) |  |
| Упакованное десятичное | 10(80) | (18 цифр) |
| Короткое вещественное | 4(32) |  |
| Длинное вещественное | 8(64) |  |
| Расширенное вещественное | 10(80) |  |

Практически во всех современных вычислительных системах минимальная адресуемая единица информации в памяти (байт) равна 8 битам. При этом возникает неоднозначность представления в памяти двоичного числа, разрядность которого больше 8 бит. Целое число можно записать в 256-ичной ( см. параграф 2.4.4) системе счисления так:

где – количество байт в представлении числа; - -й байт числа .

Итак, если нулевой байт числа хранится в памяти по адресу , первый по адресу , -й по адресу , то это порядок от младшего к старшему (а также little-endian, интеловский порядок)

Если -й байт числа хранится по адресу , -й по адресу , -й по адресу , то это порядок от старшего к младшему (а также big-endian, порядок Motorola, сетевой порядок)[[14]](#footnote-14)

Пример на рисунке Рис. 2.6.

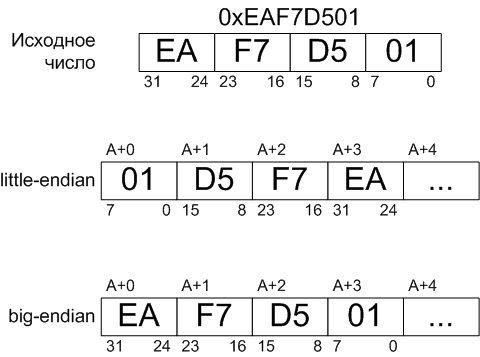


Рис. 2.6 Порядок байт для 32 разрядного числа

В некоторых микропроцессорах используется интеловский порядок (Intel x86), в некоторых сетевой (Motorola 68000, SPARC), а в некоторых – это параметр настраиваемый (ARM, DEC Alpha, MIPS, PA-RISC, IA-64[[15]](#footnote-15)).

Больший интерес представляют форматы с *плавающей точкой*, позволяющие представлять вещественные числа. Особенностью такого представления является то, что отдельно хранится информация о положении точки среди разрядов числа. При этом числа представляются в виде

где – мантисса , – экспонента.

Например. Мантисса , экспонента .

Первая двоичная цифра мантиссы всегда равна единице[[16]](#footnote-16), поэтому её можно и не хранить в формате, экономя один бит и повышая этим самым точность представления в два раза.

Кроме того, значение экспоненты хранят не в виде целого со знаком, а в виде суммы с некоторым числом так, чтобы всегда было только положительное число и чтобы вещественные числа легко сопоставлялись (сравнивались) – для этого достаточно в большинстве случаев сравнить представления экспонент. Итак, собственно форматы:

* Короткое вещественное: бит 31 - знак мантиссы, биты 30…23 – 8-битная экспонента +127, биты 22…0 – 23-битная мантисса без первой цифры (которая как известно всегда равна 1).
* Длинное вещественное: бит 63 – знак мантиссы, биты 62…52 – 11-битная экспонента +1024, биты 51…0 – 52-битная мантисса без первой цифры.
* Расширенное вещественное: бит 79 – знак мантиссы, биты 78…64 – 15-битная экспонента +16383, биты 63…0 – 64-битная мантисса с первой цифрой (вопреки и несмотря… бит 63 всегда равен 1!).

Пример представления -0.625 в коротком вещественном см. на рисунке Рис. 2.7.



Рис. 2.7 Короткое вещественное

Поднимем, наконец, вопрос «а как представить, например, ноль»? Кроме ноля возможны следующие случаи, предусмотренные стандартами IEEE:

* положительный ноль: все биты представления сброшены в ноль;
* отрицательный ноль: знаковый бит – 1, все остальные биты сброшены в ноль;
* положительная бесконечность: знаковый бит – 0, все биты мантиссы – 0, все биты экспоненты – 1;
* отрицательная бесконечность: знаковый бит - 1, все биты мантиссы – 0, все биты экспоненты – 1;
* есть и другие состояния, о которых не будем…

**Вопросы и задания:**

1. Перевести в десятичную систему счисления числа:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Вариант | 1 | 2 | 3 | 4 | 5 | 6 |
| Число |  |  |  |  |  |  |

1. Перевести из десятичной системы счисления числа в систему счисления с указанным основанием:

|  |  |  |
| --- | --- | --- |
| Вариант | Число | Основание |
| 1 | 174.875 | 2 |
| 2 | 241.33 | 2 |
| 3 | 8713.31 | 3 |
| 4 | 9715.13 | 5 |
| 5 | 11579.13 | 7 |
| 6 | 65891.31 | 8 |
| 7 | 6791501.55 | 16 |
| 8 | 6791501.55 | 17 |

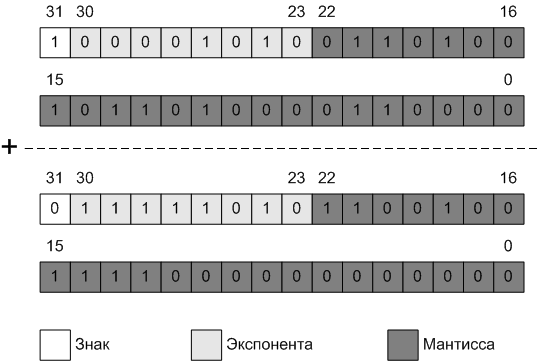
1. Перевести число из системы счисления с одним основанием в систему счисления с другим основанием:

|  |  |  |
| --- | --- | --- |
| Вариант | Число | Основание |
| 1 |  | 2 |
| 2 |  | 3 |
| 3 |  | 8 |
| 4 |  | 16 |
| 5 |  | 8 |
| 6 |  | 16 |

1. Сложить числа и в двоичной системе счисления в (размер разрядной сетки выбрать самостоятельно). Использовать дополнительный или обратный код по усмотрению преподавателя:

|  |  |  |
| --- | --- | --- |
| Вариант |  |  |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |

1. Целые числа. Используйте запись в 16-ичной системе счисления и дополнительный код.
   1. Представить -103 как байт.
   2. Представить -31317 как короткое слово (little-endian и big-endian).
   3. Представить 4568945 как слово (little-endian и big-endian).
   4. Представить -78348108 как длинное слово (little-endian и big-endian).
2. Вещественные числа. Используйте запись в 2-ичной системе счисления, пропуская разряды содержащие длинные группы одинаковых бит. Подписывайте разряды.
   1. Представить -313.625 как короткое вещественное.
   2. Представить -601.53125 как длинное вещественное.
   3. Представить 611.03125 как расширенное вещественное.
   4. Представить 0 как короткое вещественное.
   5. Представить 1 как короткое вещественное.
   6. Представить 1 как расширенное вещественное.
   7. Представить как расширенное вещественное.
   8. Представить как длинное вещественное.
3. Выполнить сложение двоичных чисел: .
4. Сложить числа в формате короткое вещественное:



**Задания повышенной сложности**

1. Головоломка. Камень весом в 40 килограмм упал и раскололся на четыре части так удачно, что с помощью его осколков теперь на весах Фемиды (чашечных) можно с точностью до килограмма взвесить груз от 0 до 40 килограмм. Каков вес осколков? Какая система счисления получилась?
2. Сложить в троичной симметричной системе счисления числа и . Цифрам соответствуют числа . Формула построения числа та же что и у рассмотренной позиционной системы счисления.

|  |  |  |
| --- | --- | --- |
| Вариант |  |  |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |

1. Перевести из десятичной системы в троичную симметричную числа:

|  |  |
| --- | --- |
| Вариант | Число |
| 1 |  |
| 2 |  |
| 3 |  |

1. Придумайте, как, имея в своем распоряжении лишь целые числа организовать сложение, вычитание, умножение и деление чисел с фиксированной запятой? Например, для представления целого числа доступно 32 бита, а мы хотим отвести 16 бит на целую часть и 16 на дробную. При этом соответствующие операции для работы с целыми числами имеются.
2. В каких случаях сложение чисел в формате с плавающей точкой приводит к потере точности?
3. В каких случаях сложение двух ненулевых чисел в формате с плавающей точкой даст результат, равный одному из операндов?
4. Дайте оценку абсолютной и относительной погрешности представления чисел в формате с плавающей запятой.

**Философия**

1. Разработать способ перевода чисел в систему счисления с основанием K, причем цифрам соответствуют числа из диапазона Привести примеры.
2. Возможно построение систем счисления, в которых цифры для -го и для -го () изменяются в различных пределах. То есть для каждого -го разряда *свое* жестко заданное основание . Целая часть получается так:

а дробная часть получается так:

где .

Например, если указано, что число представлено в системе счисления с основаниями «2|4|7|5|.|2|10|3» и дано число: 1354.191, то ему соответствует:

Разработайте обобщенный алгоритм перевода в такую систему счисления. Приведите примеры.

## Кодирование текста.

Неизвестно, о чем следовало бы сказать в первую очередь: о *числах* или о *словах*. Мы рассказали прежде о *числах* только потому, что в цифровых устройствах *текст* кодируется с их помощью. Важность *языка* в жизни людей сравнима разве что с важностью самой жизни.

В основе общения людей лежит *язык*. И, как известно, существует великое множество языков и наречий. Разделенные, большими расстояниями, временем, интересами, сверхъестественными силами[[17]](#footnote-17), или чем-то иным, группы людей стали носителями весьма не похожих друг на друга *языков*. Различные языки много заимствовали друг у друга, изменялись со временем, рождали новые, совершенствовались, упрощались, усложнялись и умирали. Язык – своеобразная память народа, самая стойкая и самая желанная к захвату и разрушению крепость в ходе информационной войны. В языке находят *отражение* объекты реального мира, действия этих объектов, отношения между ними, характеристики объектов и действий, и т.д. Находят в виде того, что было, согласно некоторым религиозным и философским воззрениям в *начале* (мира, человечества, разума, культуры,…) – *слово*.

В настоящее время создаются искусственные языки (интерлингва, эсперанто, и др.), до предела упрощенные и легкие в освоении, призванные стать инструментом глобализации, средством легкого международного общения[[18]](#footnote-18). Со временем появилась и необходимость постановки задач для управляющих узлов современных (вычислительных и не только) машин на достаточно высоком уровне. Как следствие появились языки программирования высокого уровня, структура которых приближена к естественным (людским) языкам. Их создается еще больше.

Так или иначе, *сигналом* для передачи *слова* в *начале*, похоже, был *звук*. Не очень подходящая форма для передачи описания мира потомкам. Решением стала *письменность*, которая задействовала, помимо слуха, еще один из самых *широких* каналов восприятия – *зрение[[19]](#footnote-19)*. Целые *слова* обозначались символичными рисунками – *пиктограммами*[[20]](#footnote-20). В иероглифическом письме одним составным рисованным символом – *иероглифом* может обозначаться целое понятие вкупе с его свойствами (锎). И этот способ до сих пор очень важен: мы весьма часто используем символы, служащие обозначениями понятий, как глубокой древности (☯,☥,卍,☦), так и весьма современных (☮), служащие самым разным целям (геральдическим ☭; экономическим $, €, ¥; музыкальным ♩, ♫; игровым ♔, ♕, ♖, ♗, ♘, ♙, ♠, ♡, ♢, ♣; астрологическим ♀, ♁, ♂; предупредительным ☢, ☣; эмоциональным ☺, ☹ и т.д. ). Другой подход к записи *слова*, по-видимому, связан с несовершенством голосового и слухового аппаратов человека. Так или иначе, для воспроизведения слова звуком, использовался конечный набор «кодовых» звуков (фонем). Эти звуки и стали кодироваться при письме с помощью букв[[21]](#footnote-21) – символов определенного начертания, изображений, пиктограмм. Набор букв для письма и стал называться *алфавитом* (из греческого, по первым двум буквам греческого алфавита альфа () и бета ()). Рассмотрим некоторые способы кодирования текста.

### Азбука Морзе

Азбука Морзе (Морзянка) предназначена для передачи текста. Морзянку до сих пор можно услышать на длинных волнах радиовещания. Короткий звук соответствует *точке* (.), длинный звук – тире (-). Короткие промежутки между звуками разделяют *точки* и *тире* в *символе* алфавита, а длинные промежутки разделяют *символы[[22]](#footnote-22)*. Морзянка удобна тем, что её можно передавать практически любым *сигналом*: например, вспышками света (что, например, используется на флоте), импульсами напряжения, звуком и т.д. Эта особенность часто используется в экстремальных ситуациях, например, вы никогда не докричитесь до соседа с нижнего этажа, но легко достучитесь до него по батарее, потому как проводящие звук среды будут в данном случае использоваться разные. Широко известный сигнал SOS (Save Our Souls – Спасите наши души) предназначен именно для таких ситуаций: «···---···» (три коротких, три длинных, три коротких повторяющихся бесконечно).

Схема кодирования букв такая:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Буква (ru/en) | Код Морзе | Буква (ru/en) | Код Морзе | Буква (ru/en) | Код Морзе |
| А/A | **·-** | К/K | **-·-** | Ф/F | **··-·** |
| Б/B | **-···** | Л/L | **·-··** | Х/H | **····** |
| В/W | **·--** | М/M | **--** | Ц/C | **-·-·** |
| Г/G | **--·** | Н/N | **-·** | Ч/ | **---·** |
| Д/D | **-··** | О/O | **---** | Ш/ | **----** |
| Е,Ё/E | **·** | П/P | **·--·** | Щ/Q | **--·-** |
| Ж/V | **···-** | Р/R | **·-·** | Ъ,Ь/X | **-··-** |
| З/Z | **--··** | С/S | **···** | Ы/Y | **-·--** |
| И/I | **··** | Т/T | **-** | Ю/ | **··--** |
| Й/J | **·---** | У/U | **··-** | Я/ | **·-·-** |
|  |  |  |  | Э/ | **··-··** |

Прописные и строчные буквы не разделяются. Обратите внимание, что кодирование неравномерно: для часто встречающихся букв используются более короткие коды Морзе. На самом деле, коды соответствуют вероятностному распределению букв в английской речи, а не в русской. Так как азбука Морзе активно использовалась в радиосвязи и была международной (интернациональной), то с целью упрощения запоминания буквы кириллического алфавита поставлены в схеме кодирования рядом с фонетическими близнецами из латиницы. По приведенной схеме закодировать слово просто, а вот декодировать букву по пискам из эфира весьма затруднительно. Для этой задачи удобно воспользоваться деревом кодирования (Рис. 2.8).



Рис. 2.8 Кодирование букв в азбуке Морзе

Десятичные цифры кодируются так:

|  |  |  |  |
| --- | --- | --- | --- |
| Цифра | Код Морзе | Цифра | Код Морзе |
| 0 | **-----** | 5 | **·····** |
| 1 | **·----** | 6 | **-····** |
| 2 | **··---** | 7 | **--···** |
| 3 | **···--** | 8 | **---··** |
| 4 | **····-** | 9 | **----·** |

Коды цифр достаточно легко запоминаются.

Знаки препинания представлены еще более длинными кодами:

|  |  |  |  |
| --- | --- | --- | --- |
| Цифра | Код Морзе | Цифра | Код Морзе |
| . | **·-·-·-** | ' | **·----·** |
| , | **--··--** | ( | **-·--·** |
| ? | **··--··** | ) | **-·--·-** |
| : | **---···** | = | **-···-** |
| ; | **-·-·-·** | + | **·-·-·** |
| - | **-···-** | $ | **···-··-** |
| / | **-··-·** | ¶ | **·-·-··** |
| " | **·-··-·** | \_ | **··--·-** |

Существуют и специализированные, управляющие коды, но мы ограничимся необходимым и достаточным минимумом. Увы, зубрежка – единственный способ выучить азбуку Морзе.

**Вопросы и задания**

1. -•- •- -•- •• --- -••• • --•- •- •-•• •• --••-- ••• -• •- -••• •••- •- ••-- - -••• --- --•• -•• ••- •••• --- --
2. -••• -•-- ---- • --• --- •--• -•-- --••-- -• •• •••- • - •--• •- -••• -•--
3. ••• •- -• •• -••• • --• ••- - --••-- •- --- --• •-•• --- -••• •-•• •• ••• - --- •-•- -
4. ••• -•- •- - • •--• - -••- -••• • •-•• •- -••• • ••• -••- ••• -••• • - --- -•• • •-•• •-
5. -••• • --•• •--• ••- -•- •--• •• ••• ••- • - --••-- -••• • --•• --•• ••- -••• --- -••• -•- ••- ••• •- • -
6. -• •• --• •-•• •- --•• -• •• ••- ---- • •--- --••-- •- ••• •-•• • •--• -•-- •••• -••• --- -•• •• -
7. ••- - -•- •- -••• -- --- •--• • -•••- •••• -••• --- ••• - -• •- --•• •- -••• --- •--• •
8. -•- --- -• -••- ••• - •- •-•• -••- -• --- •--- --••-- •••• -••• --- ••• - •-•• -••- -• •-•- -• --- •---
9. •-•• • - •• - -•••- -••• --- • - --••-- ••- •--• •- •-•• -•••- --•• • -- •-•• ••-- •--• --- • -
10. ---• - --- •--• --- ••• •--• • -•• -••- --•• • -- •-•• •• ••• - --- •• -

### Азбука Брайля

Предназначена азбука для слепых и слабовидящих. Позволяет не только кодировать буквы, цифры и знаки препинания, но и, например, ноты. Каждый символ в азбуке Брайля представлен знакоместом из шести точек, расположенных в два столбца и три строки. На месте одной или нескольких точек может быть выпуклость (продавленная с другой стороны). Всех комбинаций при таком двоичном кодировании может быть . Текст читается ощупью слева направо. Так как «писать» приходится с противоположной стороны листа, то пишут справа налево, отражая соответственно и точки каждого знакоместа. Пишут на плотной бумаге с помощью специального трафарета для знакомест (шеститочий) и металлического стержня – грифеля, продавливая выпуклости в заданных позициях. Далее изображения знакомест приводятся в режиме «чтение».

Кодирование букв приведено на Рис. 2.9. Перед заглавной буквой ставится «признак заглавной кириллической буквы» или «признак заглавной буквы»:

 или .



Рис. 2.9 Кодирование кириллических букв в азбуке Брайля

Пробелу соответствует пустое (не продавленное знакоместо). Знаки препинания кодируются так, как показано на Рис. 2.10. После всех знаков препинания кроме запятой, точки с запятой и дефиса должен идти пробел (указано на рисунке).



Рис. 2.10 Кодирование знаков препинания в азбуке Брайля

Перед числом, то есть группой цифр, ставится «цифровой знак»:

.

Целая часть от дробной отделяется запятой. Иногда «цифровой знак» ставят перед каждой цифрой, но это излишне.



Рис. 2.11 Кодирование десятичных цифр в азбуке Брайля

Обратите внимание, что сами коды цифр совпадают с кодами букв А(1), Б(2), Ц(3), Д(4), Е(5), Ф(6), Г(7), Х(8), И(9), Ж(0). Это опять влияние латиницы: символы кириллицы идут в том порядке, в котором расположены в латинском алфавите их фонетические близнецы (ноль незаслуженно поставлен в конец ряда чисел, а не в начало). Знаки арифметических действий кодируются так, как показано на Рис. 2.12. Перед знаками плюс, минус, разделить и равно всегда ставится пробел, а перед знаком умножения не ставится.



Рис. 2.12 Кодирование арифметических действий в азбуке Брайля

Например, выражение 20+20\*20=420 будет выглядеть так:

.

Отдельно стоит отметить, что писать азбукой Брайля весьма затруднительно, поэтому прибегают к множеству сокращений. Определенными шеститочиями и их короткими комбинациями кодируются целые (часто употребляемые) слова. Это вносит дополнительные трудности в (и без того непростую) жизнь слепых (и слепо-глухих, для которых азбука остается едва не единственным путем общения). Поэтому обычно слова в сообщениях для широкого круга, несмотря на неудобства, кодируются побуквенно, и приведенных знаний должно хватить для понимания сути.

**Вопросы и задания**

1. ⠠⠞⠑⠗⠏⠑⠝⠾⠑⠀⠊⠀⠞⠗⠥⠙⠀⠺⠎⠡⠀⠏⠑⠗⠑⠞⠗⠥⠞
2. ⠼⠃⠄⠼⠃⠀⠟⠑⠞⠮⠗⠑⠂⠀⠁⠀⠺⠀⠙⠺⠕⠊⠟⠝⠕⠍⠀⠏⠗⠑⠙⠎⠞⠁⠺⠇⠑⠝⠊⠊⠀⠼⠁⠚⠚
3. ⠠⠺⠕⠙⠁⠀⠅⠁⠍⠑⠝⠾⠀⠞⠕⠟⠊⠞
4. ⠠⠝⠑⠀⠍⠁⠇⠕⠀⠇⠊⠀⠺⠁⠎⠂⠀⠝⠑⠀⠝⠁⠙⠕⠀⠇⠊⠀⠝⠁⠎⠢⠀
5. ⠠⠁⠺⠕⠎⠾⠂⠀⠙⠁⠀⠝⠑⠃⠕⠎⠾⠀⠝⠁⠀⠺⠕⠯⠝⠑⠀⠃⠗⠕⠎⠾
6. ⠠⠺⠀⠏⠥⠎⠞⠕⠯⠀⠃⠕⠟⠅⠑⠀⠱⠥⠍⠥⠀⠍⠝⠕⠛⠕
7. ⠼⠉⠂⠁⠙⠁⠑⠊⠀⠪⠞⠕⠀⠫⠀⠵⠝⠁⠳⠀⠊⠀⠏⠕⠍⠝⠳⠀⠏⠗⠑⠅⠗⠁⠎⠝⠕

### Текст в вычислительных машинах

Далее мы рассмотрим кодирование *текста* с целью его представления в информационных системах. Неделимой единицей *текста* является *символ*. Все множество символов разделяют на *отображаемые* и *управляющие*. Отображаемым символам соответствует определенное изображение (начертание, glyph – *глиф*). Один и тот же символ (например, буква А) может быть начертан совершенно по-разному: Аа, *Аа*, Аа, Аа, Аа, Аа. Стилистика изображения символа задается *шрифтом*, который определяет отображение кода *символа* на *глиф*. *Управляющие* же символы предназначены для служебных целей. Например, у европейцев принято разбивать текст на строки, читаемые справа-налево, и располагать эти строки сверху-вниз. Встретившийся символ возврата каретки (обозначается <CR> – Carriage Return) заставляет воображаемую печатную машинку сдвинуть печатающую головку в начало строки, а символ перевода строки (обозначается <LF> – Line Feed) заставит выполнить переход на новую строку текста. К сожалению, исторически единства в вопросах перехода в начало новой строки нет. В Unix/Linux операционных системах для этого достаточно одного символа <LF>, в сетевых протоколах и в операционной системе Windows необходимы оба <CR><LF>, а в Mac OS используется <CR>. Поэтому иногда могут возникнуть проблемы с построчным отображением файла, созданным в другой операционной системе. Примером управляющего символа также может служить <BS> (Back Space), осуществлявший сдвиг печатающей головки на один шаг назад, что позволяло печатать диакритические знаки над буквами или печатать символ дважды, что давало эффект **жирного** шрифта.

Для хранения, передачи и обработки текста с помощью вычислительной машины символы требуется *закодировать*. Каждому символу ставится во взаимно-однозначное соответствие некоторое положительное целое число – код символа. Таблица кодов символов называется *кодировкой*. Эквивалентные зарубежные термины: *character encoding*, *character set* (*charset*), *character map* и *code page*. Исторически не сложилось единого стандарта кодирования символов. Так, например, для кодирования управляющих символов, символов латинского и русского алфавитов, знаков пунктуации, цифр, вполне хватает диапазона [0,255] то есть все необходимые символы можно закодировать в байте. Несмотря на то, что частота (вероятность) встречаемости символов в осмысленных текстах неравномерна, применяется (с целью упрощения декодирования) равномерное кодирование (т.е. все символы предполагаются равновероятными).

Среди байтовых кодировок (кодирующих символ одним байтом), коих развелось в свое время великое множество, наиболее известны: KOI-8r, windows-1251(cp1251), DOS(cp866). В основе указанных кодировок лежал стандарт ASCII (American Standard Code for Information Interchange) см. Рис. 2.13, который задавал соответствие основных символов числам из диапазона ([0x00,0x7F]). ASCII был разработан для нужд телеграфа, и потому он семибитный. Кодировал он символы латиницы в обоих регистрах, управляющие символы (2-3 буквенные аббревиатуры на рисунке), цифры, зраки пунктуации, разделители и некоторые специальные символы. При переходе на ЭВМ, которые хранили информацию побайтно, оставшуюся половину кодов ([0x80,0xFF]) дополнили другими символами, в том числе и кириллицей. Следует отметить, что ASCII коды были жестко фиксированы и во всех многочисленных байтовых кодировках первые 128 символов представлены единообразно. Символам остальных алфавитов в войне кодировок сильно досталось. Сравните, например, фрагменты различных кодовых таблиц для кириллицы на рисунках Рис. 2.14, Рис. 2.15. Например, русской заглавной Ю в cp1251 соответствует код 0xDE, а в KOI-8r – 0xE0.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |
| **0x0** | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | TAB | LF | VT | FF | CR | SO | SI |
| **0x1** | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| **0x2** |  | ! | " | # | $ | % | & | ' | ( | ) | \* | + | , | — | . | / |
| **0x3** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| **0x4** | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| **0x5** | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | \_ |
| **0x6** | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| **0x7** | p | q | r | s | t | u | v | w | x | y | z | { | | | } | ~ | DEL |

Рис. .13 Кодовая таблица ASCII



Рис. 2.14 Фрагмент кодовой таблицы windows-1251(cp1251). Первая половина – коды ASCII

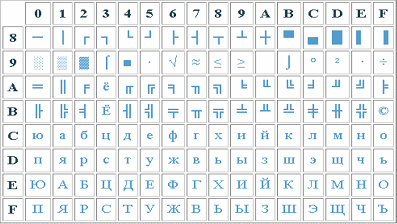


Рис. 2.15 Фрагмент кодовой таблицы KOI-8r. Первая половина – коды ASCII

Интернациональным решением является стандарт Unicode, в котором с 1991 по 1995 годы диапазон кодов был расширен до . В июле 1996 года вышел стандарт Unicode 2.0, расширивший диапазон примерно до . Это пока позволяет единообразно закодировать все алфавиты мира. На момент написания этих строк существует черновик стандарта Unicode 6.0, в котором диапазон кодов, для которого заданы соответствия символов, установлен в [0,0x10FFFF]. Например, кириллице отведено подмножество кодов [0x0400,0x04FF]. Коды, соответствующие современным кириллическим буквам приведены в следующей таблице:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |
| **0x040** |  | Ё |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **0x041** | А | Б | В | Г | Д | E | Ж | З | И | Й | К | Л | М | Н | О | П |
| **0x042** | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ъ | Ы | Ь | Э | Ю | Я |
| **0x043** | а | б | в | г | д | e | ж | з | и | й | к | л | м | н | о | п |
| **0x044** | р | с | т | у | ф | х | ц | ч | ш | щ | ъ | ы | ь | э | ю | я |
| **0x045** |  | ё |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Обратите внимание на многострадальную букву Ё – её код идет не в алфавитном порядке. Кстати, в Unicode представлена и азбука Брайля (диапазон [0x2800,0x28FF]).

Способов отображения *чисел* Unicode-кодов в конкретные *последовательности байт* несколько: UTF-8, UTF-16 и UTF-32. UTF – Unicode transformation format – способ преобразования Unicode кода в последовательность байт (неоднозначности кодирования символов числами нет! Есть лишь несколько способов представить число байтами).

Наибольшую популярность завоевывает способ UTF-8, становясь стандартом де-факто. UTF-8 отображает Unicode коды в байты неравномерно: например символы латиницы, знаки пунктуации, цифры и другие часто используемые символы кодируются одним байтом, а более редкие двумя и тремя байтами. Также стоит отметить, что латиница, цифры и знаки пунктуации занимают свое прежнее место и их представление в UTF-8, неотличимо от представления в любой другой байтовой кодировке, основанной на ACSII.

Таблица .2 UTF-8 представление кодов Unicode

|  |  |  |
| --- | --- | --- |
| Диапазон Unicode кодов символа | Последовательность байт (2 CC) в UTF-8 | Примечание |
| [0x00000000,0x0000007F] | 0xxxxxxx | ASCII см. |
| [0x00000080,0x000007FF] | 110xxxxx,10xxxxxx | кириллица, расширенная латиница, арабский, армянский, греческий, еврейский и коптский алфавит; сирийское письмо, тана, нко; МФА; некоторые знаки препинания |
| [0x00000800,0x0000FFFF] | 1110xxxx,10xxxxxx,  10xxxxxx | все другие современные формы письменности, в том числе грузинский алфавит, индийское, китайское, корейское и японское письмо; сложные знаки препинания; математические и другие специальные символы |
| [0x00010000,0x001FFFFF] | 11110xxx,10xxxxxx,  10xxxxxx,10xxxxxx | музыкальные символы, редкие китайские иероглифы, вымершие формы письменности |
| [0x00200000,0x003FFFFF] | 111110xx,10xxxxxx,  10xxxxxx,10xxxxxx,  10xxxxxx | Не используется, хотя теоретически возможно |
| [0x04010000,0x7FFFFFFF] | 1111110x,10xxxxxx,  10xxxxxx,10xxxxxx,  10xxxxxx,10xxxxxx | Не используется, хотя теоретически возможно |

Например, заглавная русская буква А (код Unicode 0x80=10000000b) в Utf-8 будет закодирована двумя байтами: <***110***00010b, ***10***000000b> = <0xC2, 0x80>. ■ Символ опасности ☠ (код Unicode 0x2620=00100110 00100000b) будет закодирован последовательностью байт: <***1110***0010b, ***10***011000b, ***10***100000b> = <0xE2, 0x98, 0xA0>. ■

Описанное кодирование текста – текстовый формат данных – используется как основа для исходных текстов практически всех известных языков программирования, языков разметки (SGML, XML) и на них в свою очередь основанных (HTML – языка разметки веб страниц, MathML – языка разметки математических формул и бесконечное множество других \*ML, RSS – языка разметки краткой сводки новостей сайта, SVG – формата векторных изображений (см. описание далее)), форматов сетевых протоколов (HTTP, FTP, SQL, SMTP, POP, и т.д.). Файл, содержащий данные в текстовом формате, называется текстовым. Программа, предназначенная для работы с текстовыми файлами, называется текстовым редактором и присутствует в базовой поставке любой операционной системы.

**Вопросы и задания**

1. Декодировать текст в cp1251: <CD E5 20 EF EE E7 E2 EE EB FF E9 20 E4 F3 F8 E5 20 EB E5 ED E8 F2 FC F1>.
2. Декодировать текст в KOI-8r: <E4 CF 20 DE C5 C7 CF 20 C4 CF D8 C5 CC 20 D0 D2 CF C7 D2 C5 D3 D3 2C 20 D4 D2 D5 C4 20 C6 C9 DA C9 DE C5 D3 CB C9 CA 20 C9 D3 DE C5 DA>.
3. Декодировать текст в UTF-8: <E2 99 83 20 2D 20 D1 8D D1 82 D0 BE 20 D1 81 D0 B8 D0 BC D0 B2 D0 BE D0 BB 20 D1 8E D0 BF D0 B8 D1 82 D0 B5 D1 80 D0 B0>.
4. Закодировать в cp1251: «Никогда ни о чем не жалейте».
5. Закодировать в KOI-8r текст: «$ – знак доллара, обозначение американских денежных знаков».
6. Закодировать в UTF-8 текст: «☢ – обозначение радиационной опасности».

## Кодирование с целью защиты свойств информации

Информационный век сделал информацию особого сорта товаром (ресурсом), который производится, продается, обменивается. Информация – это оружие в информационной войне. Информационная система – мозг любого производства вещей и услуг. Ценность информации, в большинстве случаев намного превышает ценность технических средств для её хранения, передачи и обработки. И коль информация имеет цену, то, очевидно, она не может не стать причиной конфликта. Такие юридические понятия как: государственная тайна, коммерческая тайна, интеллектуальная собственность, авторское право, ноу-хау, персональные данные и т.д. созданы с целью справедливого урегулирования конфликтов, тесно связанных с информацией. Информация может быть похищена, испорчена, подменена, уничтожена и т.д.

Выделяют следующие важнейшие (с точки зрения защиты) свойства информации: *целостность*, *конфиденциальность*, *принадлежность* и *доступность*.

Законы, конечно, позволяют защитить определенные свойства информации, но только под страхом возмездия. Впрочем, информация подвергается угрозам не только со стороны злого умысла человека, но и со стороны совершенно стихийных факторов: катаклизмов, помех, непредвиденных ситуаций. Поэтому так важны технические способы защиты свойств информации, и достойное место среди них занимают методы кодирования.

### Защита целостности информации

Как уже говорилось целостность информации – это её неизменность относительно некоторого фиксированного значения (например, это свойство дает *получателю* уверенность в том, что он получил информацию в том виде, в котором она была отправлена). В любом же канале передачи информации имеется шум, который воздействует на передаваемый сигнал, искажая его. Рассмотрим схему канала передачи информации, представленную на Рис. 2.16.

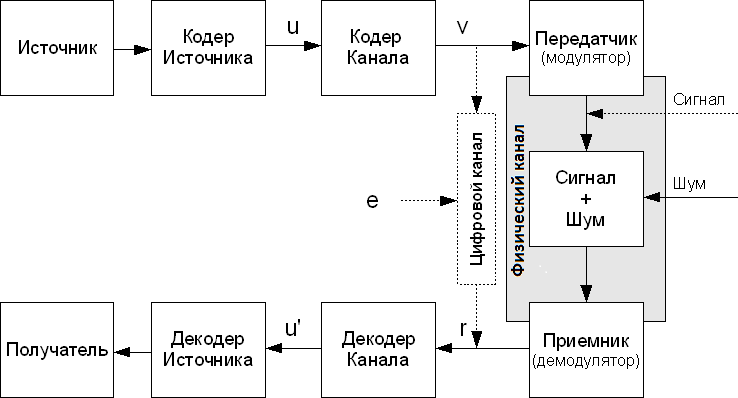


Рис. 2.16 Схема канала передачи данных

Итак, имеется источник событий (сообщений), которые, поступая на вход кодера источника, преобразуются в информацию (например, в последовательность бит, которую далее называем информационным *вектором*, или просто *вектором*). Эта информация адекватно отражает источник и не более того. Далее информационный *вектор* поступает на вход кодера канала. Кодер канала, используя определенные методики перекодирования, добавляет к информации *избыточность*, необходимую для защиты от помех и в результате выходе получается *вектор* . Так как вектор  *избыточен*, то количество информации в нем больше, чем в . Далее вектор передается на вход модулятора. Модулятор преобразует информацию в соответствующий *сигнал*, распространяющийся в некоторой физической среде. Любая физическая среда передачи сигнала (физический канал на рисунке) подвержена воздействию шумов (помех). Шум накладывается на сигнал, и результат этого наложения поступает на вход демодулятора. Если помех были несущественными, то на выходе будет вектор . Иначе, демодулятор выдаст вектор . Декодер канала, получив на входе вектор , выполнит устранение избыточности, а также постарается распознать были ли в канале ошибки и, возможно, постарается их исправить. На выходе будет получен вектор , о котором с большой уверенностью можно сказать, что он идентичен вектору . Декодер источника преобразует в события (сообщения) удобные для получателя.

*Сигнал* для передачи в физическом канале формируется модулятором из кода , поступившего от кодера канала. Совершенствуя модулятор/демодулятор (модем), повышая его качество, мы рано или поздно столкнемся с эффектом «насыщения», когда затраты велики, а прирост качества ничтожен. В этом плане совершенствование кодера/декодера (кодек) канала дает больший эффект см. Рис. 2.17. При этом мы абстрагируемся от того, что у нас вообще есть модем: выход кодека канала непосредственно «передается» в абстрактном *цифровом канале*. В этом случае цифровой аналог шума – это информационный вектор , накладывающийся на вектор .

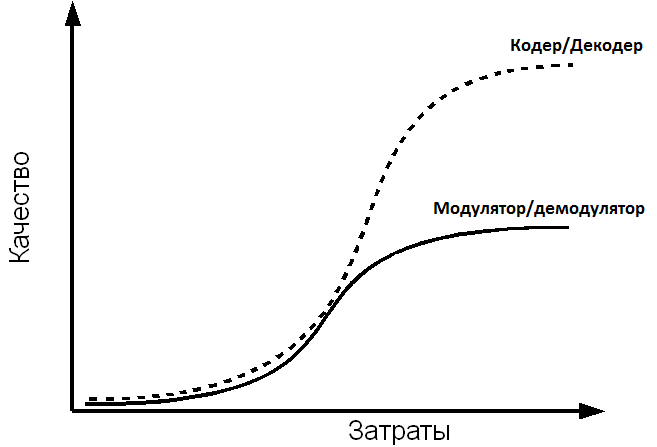


Рис. 2.17 Зависимость качества (помехоустойчивости) в зависимости от затрат на кодек канала и модем

Совершенствованием кодека мы и займемся. «Качество» цифрового канала оценивается средней долей ошибочных бит (BER - Bit Error Rate). BER определяется как средняя вероятность ошибки одного бита передаваемой информации. Чем BER меньше, тем лучше, на рисунке же Рис. 2.17 под «качеством» понималась величина обратная к BER, например, 1/BER.

Строго говоря, ошибки, возникающие в цифровом канале, могут быть следующих типов:

* замещение символа кода (в случае двоичного кодирования – инверсия двоичного разряда);
* вставка символа кода (например, добавление бита в сообщение);
* выпадение символа кода (например, исключение бита из сообщения).

Мы рассмотрим только ошибки типа замещение.

Основная идея помехоустойчивого кодирования состоит в том, чтобы внести в исходное кодовое слово дополнительные служебные биты информации, предназначенные для повышения устойчивости к помехам. В результате исходное кодовое слово будет преобразовано кодером канала в слово большей длины.

Существуют две стратегии использования вводимой информационной избыточности:

* с непосредственным обнаружением и исправлением ошибок на стороне получателя (FEC – Forvard Error Correction);
* с обнаружением ошибок и с последующим запросом на повторную передачу (ARQ – Automatic Repeat Request).

Рассмотрим ARQ стратегию на примере ИНН – идентификационного номера налогоплательщика. С этими номерами работают люди и ошибки неизбежны. Что будет, если оператор допустит ошибку при вводе такого кода? Ничего хорошего. Поэтому в ИНН есть цифры контрольного кода. Формат ИНН(различают ИНН для юридических лиц (ЮЛ) и для физических лиц (ФЛ)) представлен на рисунке Рис. 2.18. Обозначения TT – код территориального деления (номер региона), NN – номер инспекции в регионе, X – порядковый номер налогоплательщика в инспекции, C - контрольный код.

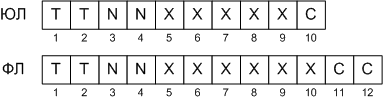


Рис. 2.18 Структура ИНН

На основе остальных разрядов ИНН формируются цифры контрольного кода.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер разряда |  |  |  |
| 1 | 2 | 7 | 3 |
| 2 | 4 | 2 | 7 |
| 3 | 10 | 4 | 2 |
| 4 | 3 | 10 | 4 |
| 5 | 5 | 3 | 10 |
| 6 | 9 | 5 | 3 |
| 7 | 4 | 9 | 5 |
| 8 | 6 | 4 | 9 |
| 9 | 8 | 6 | 4 |
| 10 |  | 8 | 6 |
| 11 |  |  | 8 |

Расчет контрольных цифр для ИНН юридического лица производится так:

где – операция получения остатка от деления числа на .

Для физического лица:

Итак, пример. Есть физическое лицо, закрепленное под номером 832630 в налоговой инспекции номер 45 в Кировской области. Сформировать корректный контрольный код для его ИНН=4345832630??.

Получаем ИНН=434583263013. Далее, допустим оператор вводил данный ИНН и ошибся, введя ИНН=. По введенным данным система вновь рассчитала контрольные коды:

Введенный контрольный код 13 и рассчитанный 00 не совпали, и система выдала оператору сообщение с просьбой перепроверить введенный ИНН.■

Еще одним примером ARQ может служить популярный контроль по четности. К исходному двоичному слову добавляется служебный бит информации (бит чётности ), содержащий сумму «по модулю два» (по исключающему или) всех бит исходного слова. Бит четности равен нулю, если количество единичных бит слова чётно, и равен единице наоборот. Также используют контроль по нечётности, при этом значение бита нечетности противоположно значению бита четности. То есть для -разрядного слова получим:

Если рассчитанный декодером после передачи бит чётности (нечётности) не совпадает со переданным значением бита четности (было получено кодером), то при передаче произошла ошибка. Причем, двойные ошибки замещения или ошибки четной кратности таким способом распознаваться не будут.

Сложение по «исключающему или» (еще говорят по xor, по «модулю два») () выполняется по правилам:

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

Перейдем к стратегии FEC. Ошибки выявляются и исправляются на принимающей стороне. Основная идея, лежащая в такой стратегии заключена в общих словах в том, что каждому исходному кодовому слову после обработки кодером канала, соответствует код с внесенной избыточностью. Необходимо отметить, что общее количество *всех возможных* кодов значительно больше, чем количество *всех возможных* кодов .



Рис. 2.19 Принцип кодирования FEC (с возможностью исправления ошибок)

Когда в коде вследствие воздействия шума ( на рисунке Рис. 2.19) инвертируется один или несколько бит, он перейдет в код . Все возможные коды отображаются на коды (одному коду соответствует несколько кодов ). При этом возможные ситуации представлены на рисунке Рис. 2.19. То есть, если вследствие ошибки будет получен вектор , который соответствует тому же вектору , что и , то декодером канала ошибка будет обнаружена и исправлена. Если вследствие ошибки будет получен вектор , который соответствует вектору , но исходный этому не соответствует, то декодером канала будет обнаружена ошибка, но исправлена она будет неверно. Ну и если исходный вектор переходит в вектор , то декодер канала ошибки, несмотря на то, что она была, не обнаружит.

Подавляющее число ошибок в современных каналах являются *одиночными*, так как современные каналы имеют достаточно низкий показатель BER. Например, для волоконно-оптических каналов BER равен приблизительно .

Рассмотрим несколько примеров кодирования для исправления одиночной ошибки.

Можно кодировать каждый бит информационного слова *тремя* битами: , а декодировать по схеме . Очевидно, что, если в трех битах кода произойдет одиночная ошибка, то будет восстановлен правильный код .

Передаётся слово: 101. Кодируется: 111000111. Поступает в канал. Возникает ошибка: . Декодируется: 101. При этом декодер «обнаруживает и исправляет» возникшую ошибку.■

Конечно, приведенный код слишком избыточен, он увеличивает нагрузку на канал передачи данных в три раза. Это недопустимо. Нужны более экономичные методы.

Метод, дающий некоторую ощутимую экономию, приведен на рисунке Рис. 2.20. Исходное слово укладывается построчно в матрицу. И для каждого столбца и строки формируются избыточные биты контроля по четности.



Рис. 2.20 FEC. Матричный контроль

Далее рассмотрим часто применяемый на практике код Хемминга, который дает максимальную экономию для FEC стратегии. Вам понадобятся знания о двоичной системе счисления, которые, если будет что-либо не понятно можно почерпнуть в разделе 2.4.3.

Рассмотрим, как сформировать код Хемминга в его классическом варианте. Код призван исправлять одиночные ошибки. Наиболее компактным будет код, который будет позволять находить *номер* разряда ошибочного бита. При этом пусть нулевой бит в таком представлении будет *фиктивным* и полученный от декодера нулевой номер ошибочного разряда будет означать, что ошибки не было. Таким образом, если необходимо передать исходное кодовое слово длиной бит, то нужно добавить к этому слову дополнительных (избыточных) бит исходя из равенства:

|  |  |
| --- | --- |
|  | (2.3) |

где – именно столько номеров бит слова можно равномерно закодировать с помощью дополнительных бит, а общая длина кода с учетом фиктивного нулевого бита.

Классический вариант кода Хемминга получается следующим образом.

1. В двоичном числе длиной бит (без фиктивного разряда) дополнительные (контрольные) бит следует разместить в разрядах с номерами, равными степени двойки (…). бит слова следует разместить в оставшихся разрядах.
2. Каждый контрольный бит в разряде получается как сумма по модулю два («исключающему или») бит кода, находящихся в разрядах с номерами, двоичное представление которых содержит единицу в разряде (кроме самого контрольного, конечно).

При декодировании контрольные разряды пересчитываются и складываются с соответствующими контрольными разрядами в полученном . В результате в контрольных разрядах будет получено число – двоичное представление номера разряда ошибочного бита в векторе .

Например, нужно передать по каналу, в котором возможны одиночные ошибки.

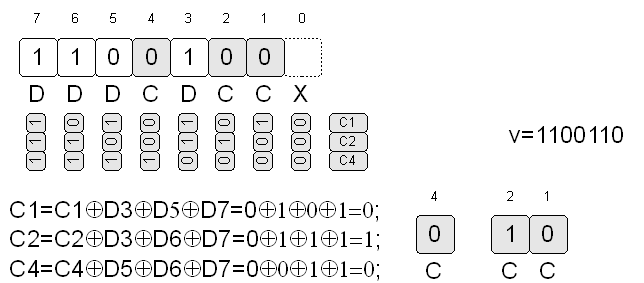


Рис. 2.21 FEC. Кодирование по Хеммингу

Итак, подбираем по формуле (2.3) минимально возможное . Кодирование и декодирование показано на рисунках Рис. 2.21 и Рис. 2.22 соответственно.

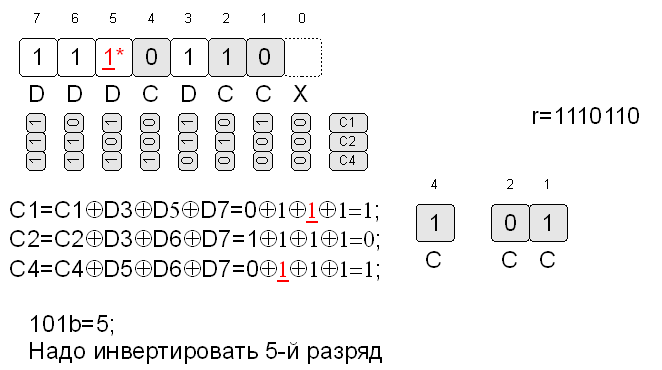


Рис. 2.22 FEC. Декодирование по Хеммингу

Поясним, например, как формируется . Нужно сложить по модулю два все биты в коде, находящихся в разрядах, двоичные представления номеров которых содержат в нулевом разряде единичные биты. Это позиции 3, 5 и 7 (1 - позиция контрольного кода и её в сумму не включаем).

Код Хемминга компактен, например, для 32 разрядного двоичного слова потребуется всего 6 дополнительных контрольных разрядов.

Избыточная информация, как правило, называется *контрольной суммой*. Способов получения контрольных сумм много, и кроме рассмотренных вариантов, лишь вскользь упомянем наиболее популярные способы контроля целостности: циклический контроль (CRC16, CRC32), коды Рида-Соломона, дайджесты md4, md5, sha… Например, в сети интернет очень часто для скачиваемых программ выкладываются отдельно их контрольные суммы md5 для того, чтобы выполнить проверку целостности скачанной программы.

**Вопросы и задания**

1. Корректен ли ИНН физического лица 434573243018?
2. Сформируйте ИНН физического лица, имеющего порядковый номер 945234 в 45 налоговой инспекции города Кирова.
3. Сформируйте ИНН юридического лица, имеющего порядковый номер 14523 в 45 налоговой инспекции города Кирова.
4. Сформируйте код четности (и нечетности дополнительно) для приведенных двоичных слов: <1001010b, 11110011b, 11001101b>.
5. Сформируйте матричный код для двоичных чисел заданной разрядности:

|  |  |
| --- | --- |
| Вариант | Разрядность |
| 1 | 4 |
| 2 | 5 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 11 |

Приведите примеры кодирования и исправления ошибки при декодировании.

1. Сформируйте код Хемминга для двоичных чисел заданной разрядности:

|  |  |
| --- | --- |
| Вариант | Разрядность |
| 1 | 4 |
| 2 | 5 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 11 |

Приведите примеры кодирования и исправления ошибки при декодировании.

1. Получен вектор (см. варианты заданий), причем известно, что этот вектор – результат кодирования по Хаффману вектора разрядностью 11 бит. Что можно сказать о том, как прошла передача?

|  |  |
| --- | --- |
| Вариант |  |
| 1 | 101110101010101 |
| 2 | 111011001110111 |
| 3 | 101110101010011 |
| 4 | 101011101011100 |
| 5 | 111000101000001 |
| 6 | 101100001010111 |

1. В каких разрядах для приведенного примера кодирования по Хаффману должны возникнуть двойные ошибки, чтобы декодер канала «исправил» бы 3 разряд.
2. Приведенный алгоритм Хаффмана не гарантирует обнаружения двойных ошибок – они исправляются как одиночные. Дополните метод кодирования так, чтобы двойные ошибки выявлялись (но не исправлялись).

### Защита конфиденциальности информации

*Конфиденциальность* – это, как уже известно, недоступность третьим лицам. Первые два лица – это *источник* (*отправитель*) и *получатель* информации в процессе передачи информации. Третье лицо – злоумышленник, для которого передаваемая информация должна оставаться недоступной. Выход – ограничить доступ третьих лиц к каналу передачи информации, либо перекодировать информацию так, чтобы её восприятие стало им недоступно. Такое перекодирование называется *шифрованием*. Об особенностях *шифрования* и поговорим далее.

Разработкой методов *шифрования* (в том числе) занимается *криптография*. Исследованием творений *криптографии* на стойкость занимается *криптоанализ*. Оба направления, *криптография* и *криптоанализ*, объединяются в единую науку – *криптологию*.

Методы шифрования разделяют на два вида: *симметричные* и *асимметричные*. Очень важным условием для существования самой возможности шифрования является наличие *секрета*.

На рисунке Рис. 2.23 приведена схема организации канала передачи конфиденциальной информации, на основе *симметричного* шифрования.



Рис. 2.23 Канал передачи конфиденциальной информации на основе симметричного шифрования

Исходное сообщение (информация M) отправитель преобразует с помощью шифратора в *шифротекст* S. Для этого преобразования шифратором используется функция , вторым аргументом которой является секретная информация – *ключ* K. Ключ K известен только отправителю и получателю, они и только они разделяют этот *секрет*. Злоумышленник, который в данной схеме назван *криптоаналитиком*, секретом K овладеть не может по определению! С выхода шифратора шифротекст S передается через *открытый канал* получателю. При этом *шифротекст* может получить и *криптоаналитик*. Более того, *криптоаналитик* даже может внести изменения в *шифротекст*, и до получателя дойдет уже не S, а S'. Получатель восстанавливает из шифротекста S' исходное сообщение M' с помощью *дешифратора*. *Дешифратором*, используется функция , позволяющая выполнить обратное преобразование: . Без знания *ключа* восстановить из *шифротекста* S исходное сообщение M практически невозможно[[23]](#footnote-23). Даже зная функции и , *криптоаналитик*, не зная ключа K, не в силах восстановить сообщение. В этом, к слову, и заключается важнейшее правило криптографии, сформулированное голландским криптологом Керкхоффом: стойкость шифра должна зависеть *только* от секретности ключа.

Криптоаналитик может испортить жизнь законным участникам обмена, исказив или подменив шифротекст. Увы, осмысленных изменений без знания ключа ему не сделать! Более того, если до шифрования в сообщение M была внесена избыточность с целью защиты *целостности*, то факт вмешательства будет обнаружен получателем после *дешифрования* и заработает стратегия ARQ.

Следует обратить внимание, что в схеме используется *секретный* канал. Спрашивается, раз есть секретный канал для передачи ключей, то почему же не использовать это сокровище для передачи сообщений? Увы, секретный канал слишком дорог[[24]](#footnote-24). Использовав его один раз для обмена секретным ключом, законные отправитель и получатель теперь могут организовать секретный канал передачи на основе любого *открытого*, что гораздо дешевле.

*Симметричным* шифр назван потому, что один и тот же ключ применяется как для шифрования, так и для дешифрования.

Существует множество симметричных шифров. Современный симметричный шифр стоек и математически обоснован. Стандартом в настоящее время является шифр AES, в девичестве Rijndael. Активно используется и модификация бывшего стандарта – шифр DES3. Впрочем, помимо стандартов существует множество проверенных временем шифров, которые могут использоваться в системах защиты информации. Использовать недавно созданный *гениальным* автором *сверхстойкий* шифр нужно лишь в целях информационного самоубийства.

В качестве примеров симметричных шифров мы приведем давно вымершие шифры, так как они просты для понимания.

Шифр цезаря представлял собой циклический сдвиг алфавита на одну букву вперед. Впрочем, щйхспн ёдп обигбуэ оёмэиа. Шифр простой замены выполнял взаимно-однозначную замену букв текста из одного алфавита, числами или буквами другого (или того же) алфавита. Ключом была таблица замен. Но так как вхождения букв в осмысленном тексте не равновероятны (например, буква А, или пробел встречаются в любом тексте чаще других), а вхождения пар или троек букв еще более закономерны, то такой шифр вскрывался очень легко. Шифры полиалфавитной замены, выполняли эту замену так, что все символы шифротекста присутствовали примерно в равных количествах, и частотный анализ текста не давал результатов. Некоторые из таких шифров продержались века, прежде чем были вскрыты с помощью компьютера. Шифры перестановки по определенному правилу (ключу) переставляли буквы исходного текста. Пример такого шифра (решетка) представлен на рисунке Рис. 2.24.



Рис. 2.24 Пример симметричного шифра перестановки

Канал передачи конфиденциальной информации на основе *ассиметричного* шифрования представлен на рисунке Рис. 2.25.



Рис. 2.25 Канал передачи конфиденциальной информации на основе асимметричного шифрования

Особенность *ассиметричного* шифра заключается в том, что для шифрования и дешифрования используются *разные* ключи (sk, pk). *Пара* ключей <sk, pk> генерируется *получателем* относительно легко. Один из этой пары ключей будет *открытым* pk (public key), и может быть передан *неограниченному* числу отправителей. Ключ sk, парный открытому, держится *отправителем* в *секрете*: sk – secret(private) key. Важнейшей особенностью схемы является то, что, зная открытый ключ pk, вычислить по нему *секретный* sk практически невозможно. Шифрование осуществляется с помощью открытого ключа, дешифрование с помощью секретного. Более того, восстановить из шифротекста исходное сообщение без знания секретного ключа практически невозможно: G(F(M, pk), sk)=M. Криптоаналитик может завладеть открытым ключом (впрочем великодушный получатель может сразу послать ему копию), но дешифровать сообщения он не сможет. Ему остается только слать получателю зашифрованные проклятия. Расшифровать сообщения может только обладатель *секрета* sk.

Внимательный читатель обратил внимание на то, что открытый ключ pk передается по *аутентичному* каналу. Да, *открытый* канал для передачи *открытого ключа* pk нельзя использовать ни в коем случае! Представте себе, что злоумышленник сгенерировал свою пару <sk1, pk1>, перехватил pk, когда он передавался отправителю, и вместо перехваченного ключа послал отправителю от имени получателя свой открытый ключ pk1. Отправитель, будучи уверен в том, что pk1 – это открытый ключ получателя, зашифрует для него сообщение. Злоумышленник перехватит его, расшифрует на своем sk1, внесет необходимые поправки, зашифрует перехваченным pk и отошлет получателю. Отправитель и получатель отныне общаются через двуликого Януса[[25]](#footnote-25). Как же быть? Нужно использовать *аутентичный* канал. Аутентичный канал гарантирует *целостность* и *принадлежность* информации. Получив данные по такому каналу, вы уверены в том, что информация в процессе передачи не была испорчена (*целостность*), и в том, что вы получили её именно от того, от кого и рассчитывали получить (*принадлежность*). *Аутентичный* канал дороже *открытого*, но дешевле *секретного*. Можно использовать *секретный* канал вместо *аутентичного*, хуже не будет, а в ряде случаев это просто необходимо.

В параграфе 2.6.3 мы разберемся, как гарантировать *принадлежность*.

Распространенные и проверенные временем схемы ассиметричного шифрования (например RSA, алгоритмы на основе эллиптических кривых ECES) весьма сложны, мы рассмотрим один простейший пример такой схемы: метод Диффи-Хеллмана.

Метод Дифффи-Хеллмана основан на весьма стойкой задаче *дискретного логарифмирования*. Дело в том, что в выражении , вычисляется очень быстро (есть алгоритмы быстрого возведения в степень по модулю[[26]](#footnote-26) ), а вот найти по известному – задача очень непростая при больших . Итак, собственно метод Диффи-Хеллмана.

Предварительно абоненты обмениваются по *аутентичному* каналу некоторыми параметрами, которые могут быть доступны всем желающим. Итак, выбирается большое простое (512 и более бит[[27]](#footnote-27)) число . При этом дополнительно накладываются ограничение: разложение на простые[[28]](#footnote-28) сомножители должно содержать, по крайней мере, один большой простой сомножитель. Также выбирается образующий[[29]](#footnote-29) элемент .

Параметры , – общеизвестны. Абоненты случайным образом генерируют *секретные* ключи: абонент генерирует , а абонет генерирует . Секретный ключ известен только тому, кто его сгенерировал. Далее абоненты вычисляют соответствующие своим секретным ключам, *открытые* ключи: абонент вычисляет , а абонет B вычисляет . Открытые ключи , также общедоступны по *аутентичному* каналу.

Когда абоненты желают обменяться секретным сообщением (чаще всего это число-ключ, которое будет использовано для шифрования будущих сообщений, так называемый сеансовый ключ) они могут зашифровать его на фактически уже имеющемся у них «общем» ключе .

Восстанавливается этот «общий» ключ следующим образом.

Абонент знает параметры , , открытый ключ абонента – и собственный секретный ключ , он вычисляет

Абонент знает параметры , , открытый ключ абонента – и собственный секретный ключ , он вычисляет

Но

Никто, кроме абонентов и сгенерировать не может. Так как для этого потребуется решить задачу *дискретного логарифмирования*. ■

Чтобы разобрать пример, придется научиться выполнять быстрое возведение в степень по модулю : .

Можно просто выполнить: , но это займет слишком много времени при больших целых . Представим в двоичной системе счисления:

Тогда:

Соответственно -е сомножимое равно 1, когда двоичный разряд . Введем обозначение :

При этом

Заметим, что:

Итак, рекурсивное правило:

…

Алгоритм быстрого возведения в степень.

1. Находится двоичное представление числа : . Устанавливается .
2. Находится . Далее , , .
3. Если то найдено, иначе переходим к пункту 2. ■

Пример. Найти . Первым делом находится двоичное представление . Устанавливается . А далее шаги 2-3 сведены в таблицу:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 0 |  | 1 | 1\*89 mod 613 = 89 |
| 1 |  | 0 | 89\*1 mod 613 = 89 |
| 2 |  | 0 | 89\*1 mod 613 = 89 |
| 3 |  | 0 | 89\*1 mod 613 = 89 |
| 4 |  | 1 | 89\*537 mod 613 = 592 |
| 5 |  | 1 | 592\*259 mod 613 = 78 |
| 6 |  | 0 | 78\*1 mod 613 = 78 |
| 7 |  | 1 | 78\*427 mod 613 = 204 |

Ответ: . Вместо 177 последовательных умножений по модулю обошлись шестнадцатью. В общем случае выигрыш по количеству умножений можно оценить как .■

Стоит также заметить, что *асимметричные* алгоритмы работают гораздо *медленнее* симметричных, так как требуют больших затрат на вычисления. Шифровать большие объемы информации ассиметричным шифром на практике очень неудобно. Для шифрования мультимедиа данных (аудио, видео-потоков) больного объема в реальном времени асимметричная схема просто неприменима – для этих целей используются только симметричные шифры. Однако и отказываться от столь замечательных свойств ассиметричной схемы нельзя. Выходят из положения так: генерируют случайным образом *сеансовый* ключ K для симметричного шифра. На этом ключе шифруется (симметричным шифром) передаваемое сообщение M. Далее асимметричным шифром шифруется ключ K симметричной схемы и добавляется в конец зашифрованного сообщения M. Процесс дешифрования вполне очевиден. Сеансовый ключ как явствует из названия, нужен лишь для одного сеанса связи. Настоятельно не рекомендуется использовать один и тот же сеансовый ключ в разных сеансах связи.

Напоследок следует сказать несколько слов об *идеальном* шифре. Точнее об *идеально стойком* шифре. Существует ли такой? Существует! Это шифровальный блокнот Вернама. Шифр симметричный. Как им пользоваться? В обстановке строгой секретности изготавливается два идентичных блокнота. На каждом листке такого блокнота напечатана одна буква (можно распечатать в блокноте, например, выбранный наугад абзац из любимой книжки или случайные буквы). Первая буква сообщения «складывается» с буквой на первой страничке блокнота. На самом деле, конечно, складываются порядковые номера букв в алфавите, и от полученной суммы находится остаток от деления на общее количество букв в алфавите. Это и будет номер буквы шифротекста. А+А=А (номера с нуля!), Б+Б=В, В+Г=Е, Я+А=Я, Я+Б=А… Первую букву шифротекста выписываем на бумажку, а первую страничку блокнота съедаем. И так далее. В процессе дешифрования в том же порядке съедается и второй блокнот. При этом буквы со страниц блокнота «вычитаются» из шифротекста. Аналогом шифра Вернама в двоичных вычислительных машинах будет сложение битового представления текста M с битовым представлением ключа K по «модулю два»: . См. таблицу для в параграфе 2.6.1. Дешифрование заключается в сложении шифротекста S с ключом K: . *Ключ должен быть уничтожен*. В обоих случаях видно, что длина ключа равна длине сообщения. И очевидно, что вместо того, чтобы подбирать ключ, *криптоаналитику* лучше сесть и сочинить свое сообщение подходящей длины. На том и успокоиться.

Современные шифры не *идеальны*. Длина ключа гораздо меньше длины сообщения. Поэтому подбор ключа имеет смысл. Но возможных вариантов ключа такое огромное количество, что требуется очень большое время, чтобы подобрать подходящий. Жалких 100 лет на вычисления, и криптоаналитик потеряет интерес к задаче. Впрочем, информация обычно теряет актуальность гораздо быстрее, чем люди. Современные шифры являются *вычислительно* стойкими. И когда вычислитель станет выполнять перебор ***в два*** раза быстрее, теоретически достаточно увеличить длину ключа ***на один*** бит. Поэтому везде, где упоминалось (или еще будет упоминаться), что криптоаналитику *невозможно* обойти защиту, следует смягчить формулировку до *вычислительно невозможно*.

### Защита принадлежности информации

Принадлежность – свойство информации, позволяющее определить её *источник* илифакт её получения *приемником*. Сосредоточимся пока на задаче определения *источника*. При этом *источник* (отправитель в предыдущих схемах) должен доказать, что информацию послал именно он, а не верблюд[[30]](#footnote-30), а получатель убедиться, что, действительно, не верблюд. Итак, отправитель должен дополнить свое сообщение избыточной информацией так, чтобы убедить получателя, что информацию мог отправить только он. Как это сделать, если клятвам никто не верит? Отправителю нужно *доказать*, что он *владеет некоторым секретом*, которым может владеть только он, и притом не раскрыть этот *секрет* остальным участникам обмена.

Простейший случай: отправитель и получатель уже разделяют секрет. То есть когда-то по секретному каналу связи договорились о *секретном ключе* K. Оба уверены, что секрет знают только они и никто другой. Они могут просто шифровать сообщения с помощью симметричного шифра. И, если шифруемые сообщения содержат контроль целостности, то можно быть спокойным. Но порой шифровать сообщения (сохранять конфиденциальность) нет необходимости, требуется лишь доказать что сообщение не сфабриковано.

Одним из важнейших инструментов для решения этой задачи является *криптографическая хеш-функция*. Отличия *криптографической* хеш-функции от обычной заключаются в том, что эта функция обладает *стойкостью к* *коллизии* первого и второго рода. Обычная хеш-функция H преобразует информационную последовательность символов M (в частном случае бит) произвольной длины в последовательность h фиксированной длины . Вычисляется хеш-функция очень быстро и обычно используется для ускорения поиска символьных имен в таблицах переменных при компиляции/интерпретации или для проверки целостности информации (как контрольная сумма).

Количество всех возможных аргументов M функции H огромно (так как ограничений на длину M нет, то можно сказать, что количество всех аргументов бесконечно). Количество же значений h конечно и относительно невелико. В случае кодирования битами , где длина битовой цепочки на выходе хеш-функции. Таким образом, на одно и то же значение h отображается огромное множество аргументов M. Сей факт называется *коллизией,* когда , но .

Криптографическая хеш-функция H *необратима*: вычислительно невозможно найти для заданного значения h хеш-функции H её аргумент M, такой, что H(M)=h.

Как было сказано, криптографическая хеш-функция обладает стойкостью следующих родов.

* Стойкость к коллизиям первого рода: для заданного сообщения вычислительно невозможно подобрать сообщение , такое что .
* Стойкость к коллизиям второго рода: вычислительно невозможно сформировать пару сообщений , таких что .

Итак, способ доказать принадлежность сообщения M отправителю при разделении отправителем и получателем секрета K, заключается в следующем. Отправитель находит значение h криптографической хеш-функции от конкатенации (соединения) сообщения M и секрета K: h=H(MK). Получателю передается конкатенация сообщения M и полученной контрольной суммы: Mh. Получатель, разделив M и h, вычисляет ту же хеш-функцию от конкатенации полученного M и имеющегося у него ключа K: H(MK). Если вычисленное значение H(MK) совпадает с полученным h, то можно быть уверенным в том, что сообщение послано отправителем, так как кроме получателя только он знает секретный K. Иначе сообщению M верить нельзя. Значение h называется кодом *аутентичности* или MAC (Message authentication code). Совпадение рассчитанного и переданного MAC дает гарантии и *целостности* и *принадлежности* сообщения M.

С какими проблемами сталкивается злоумышленник, желая подделать подпись MAC? Восстановить по значению H(MK) секретный ключ K он не может, хотя бы потому, что H *необратима*. Даже восстановление (что вычислительно невозможно из-за стойкости первого рода), такого, что ничего не даст, так как количество вариантов огромно.

Другой подход, построенный на *ассиметричной схеме*, гарантирующий *принадлежность* называется *цифровой подписью*. Кстати, федеральным законом об электронной цифровой подписи, принятом в 2002 году, установлен юридический статус этому понятию. Цифровая подпись является доказательством в суде, как и рукописная подпись под бумажным документом. Рассмотрим схему передачи цифровой подписи, представленную на рисунке Рис. 2.26.



Рис. 2.26 Схема передачи цифровой подписи

Как видно, схема очень похожа на схему ассиметричного шифрования (Рис. 2.25). Все сказанное в отношении схемы шифрования справедливо и для цифровой подписи, за исключением того, что обладателем секрета теперь является отправитель, и шифрование *слепка сообщения* h производится на секретном ключе sk. *Цифровой подписью* называется результат шифрования слепка – DS. Без знания секретного ключа sk шифрование невозможно. Любой желающий, имеющий открытый ключ pk, может дешифровать сообщение (проверить подпись). Итак, зашифровать *слепок* может только один *отправитель*, а расшифровать – любой желающий проверить подпись *получатель*.

При этом получатель, расшифровывающий DS открытым ключом pk, уверен в том, что зашифровать *слепок* мог только обладатель *секретного* ключа sk (парного открытому ключу pk). Уверен потому, что получил открытый ключ по *аутентичному* каналу.

Впрочем, пока не ясно, что представляет собой *слепок* h сообщения M. Слепок может представлять собой копию сообщения M. Обычно и сообщение M, и его цифровая подпись DS передаются одним блоком <M,DS>. Получателю достаточно расшифровать цифровую подпись DS и сравнить с M. Совпадение будет означать, что полученное M *аутентично*, так как гарантированы его целостность и принадлежность отправителю. Злоумышленник не сможет внести идентичные изменения в сообщение и в подпись, так как должен для этого вновь зашифровать измененную копию M, а это вычислительно невозможно без обладания секретным ключом sk. На практике (в целях экономии памяти) слепок h формируется как результат хеширования сообщения с помощью криптографической хеш-функции: h=H(M). Аналогично, сообщение M и его цифровая подпись DS=F(H(M), sk) передаются одним блоком <M,DS>. Получатель, разделив, блок на части M' и DS', находит H(M') и сравнивает с расшифрованной подписью h'=G(DS', pk). Если эти величины равны – сообщение M *аутентично*. Злоумышленник, подделывая подпись, столкнется с необходимостью, либо зашифровать вычисленный им слепок H(M') измененного сообщения M' (что вычислительно невозможно без знания sk), либо подобрать такое осмысленное сообщение M', чтобы оно давало такой же хеш, что и исходное M (что вычислительно невозможно, так как H обладает *стойкостью к коллизиям первого рода*).

Мы не будем рассматривать методы конкретные алгоритмы цифровой подписи, ограничившись описанием принципов. Интересующиеся могут разобрать методы RSA или Эль-Гамаля. Или изучить современные стандарты цифровых подписей: DSA (основана на вычислительной стойкости разложения числа на простые сомножители), ECDSA (основана на вычислениях на эллиптической кривой).

Вопросы

1. Что значит высказывание «Аутентичное сообщение»?
2. В чем разница между симметричным и асимметричным шифром?
3. В чем разница между секретным, открытым и аутентичным каналом?
4. Чтобы организовать аутентичный канал на основе открытого, вам достаточно поставить электронную цифровую подпись под сообщением. Но для того, чтобы начать использовать цифровую подпись требуется аутентичный канал. Так как его организовать?
5. Что такое *криптографическая* хеш-функция?
6. Что такое сеансовый ключ?
7. С какими проблемами сталкивается криптоаналитик, когда имеет дело с симметричным шифром.
8. С какими проблемами сталкивается криптоаналитик, когда имеет дело с асимметричным шифром.
9. С какими проблемами сталкивается криптоаналитик, когда имеет дело с MAC.

Практические задания

1. Восстановите сообщение из шифротекста. Шифр простой замены. Речь в сообщении идет о криптографии и стеганографии: <09 14 23 20 28 29 10 29 27 14 23 20 14 03 20 27 24 23 13 23 24 26 12 08 29 29 14 03 18 26 11 09 12 02 07 27 15 14 08 12 18 20 14 16 27 26 27 00 12 10 29 14 03 23 23 01 07 27 13 29 21 14 18 26 29 16 20 23 24 26 12 08 29 21 14 03 18 26 11 09 12 27 20 14 03 32 11 03 28 14 03 23 23 01 07 27 13 29 21>.
2. Восстановите сообщение из шифротекста. Шифр простой замены. Речь в сообщении идет о дешифровании: <11 00 03 08 22 28 14 22 07 25 30 32 11 29 11 22 00 03 21 28 21 22 32 11 05 08 31 04 21 28 25 07 08 11 22 28 22 01 29 21 12 22 00 21 21 24 27 11 07 08 08 22 29 21 22 28 14 22 11 15 21 22 03 11 15 26 21 22 32 11 05 08 31 04 18 11 29 11 22 08 22 13 21 30 12 11 29 11 22 11 15 21 22 00 12 14 00 03>.
3. Восстановите сообщение из шифротекста. Шифр моноалфавитной замены.

|  |
| --- |
| кч\_ча":.псыезц:тич:купр  высыаым:оибвпа:ыипр  чвдлид\_пг:кбхчлцпа"  ц:узыисдтзд:уы.чицпа"  здезч:ицкхбп:зсц\_па  ч:у:тичукчуаыезым:тыпа  тич:епуахыл:тич:купл:хциы\_пй  ,кы:уц\_д:сч:к:ыоыиы\_п,н |

1. Восстановите сообщение из шифротекста. Шифр – решетка (см. Рис. 2.24).

|  |
| --- |
| **алам**  **а"м**  **у"м**  **амыр** |

1. Восстановите сообщение из шифротекста. Шифр – решетка (см. Рис. 2.24). Подсказка: в сообщение присутствует слово «поблажку». Регистр имеет значение

|  |
| --- |
| **К отебльлб**  **яаж бекту**  **зОдна а ж**  **птьа оОлос**  **с воселйбо**  **вжтдиздае**  **уднм аюя с**  **юоо рвтерт**  **е шуба ь ш**  **кпраоу бСо** |

1. Найти

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариант |  |  |  | (для проверки) |
| 1 | 109 | 225 | 613 | 512 |
| 2 | 109 | 225 | 487 | 195 |
| 3 | 89 | 177 | 443 | 311 |
| 4 | 97 | 153 | 257 | 43 |
| 5 | 113 | 209 | 257 | 221 |
| 6 | 127 | 201 | 443 | 385 |

1. Абоненты и используют метод Диффи-Хеллмана. Они сгенерировали по секретному ключу , соответственно. Найдете общий ключ, если заданы следующие параметры и .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариант |  |  |  |  |
| 1 | 131 | 200 | 197 | 281 |
| 2 | 217 | 301 | 272 | 313 |
| 3 | 138 | 62 | 53 | 359 |
| 4 | 378 | 395 | 341 | 401 |
| 5 | 117 | 500 | 111 | 503 |
| 6 | 200 | 618 | 65 | 1021 |

## Кодирование данных

Термин *данные* столь же неоднозначен, что и термин *информация*. Волевым усилием мы приняли одно из определений информации в самом начале: Определение 1.1 на странице 1. Допустим еще одну вольность и дадим *удобное* для нас определение *данных*.

Пусть, у нас есть восемь *байт* информации. Носителем каких *данных* является эта *информация*? Может быть это географические координаты – пара коротких вещественных чисел (4 байта см. Таблица 2.1) <широта, долгота>? А может это короткий текст из восьми символов в одной из байтовых кодировок? Или может быть, что эти восемь байт – это пара коротких вещественных, представляющих действительную и мнимую компоненты комплексного числа ? Может это одно число: длинное слово или длинное вещественное? А может это закодированные события? Шифр? Ключ шифра? Программа? Может… Увы, это шесть байт информации и не более того. Мы пытались *интерпретировать* (в данном случае *декодировать*) эту информацию, пытались понять, что нашло отражение в этой цепочке символов-бит.

*Информация*, для которой известно как её *интерпретировать*, называется *данными*.



Рис. 2.27 Информация и данные

Для того чтобы *интерпретировать* информацию нужно знать *структуру* или *формат* *данных* ею представленных. Как определяют *форматы данных*?

Иногда данные определяют как набор *переменных*. Это определение ближе программистам, так как в любом языке программирования есть средства описания *структур данных*. Переменная обычно характеризуется *именем* и *значением*. *Значению* переменной соответствует определенный фрагмент интерпретируемой информации. С *именем* обычно связывают описание, проясняющее назначение переменной. *Значением* переменной являются в свою очередь *данные* определенного *формата* (или, как говорят программисты, *типа*). Существуют базовые *типы данных*: целые или вещественные числа, байты, символы текста, строки символов, и т.д., ну и конечно неделимые *кодовые символы* (биты в случае двоичного кодирования).

Бывает, что подобного определения *данных* бывает недостаточно. Тогда данные представляют собой *выражения* на некотором *формальном языке*. Тема *формальных языков* очень обширна, и обсуждать её детально не будем. Но мы приведем несколько примеров языков: язык разметки текстовых данных XML, язык регулярных выражений, язык логической разметки документов. Также мы представим несколько программ (программа это также данные) на некоторых языках программирования.



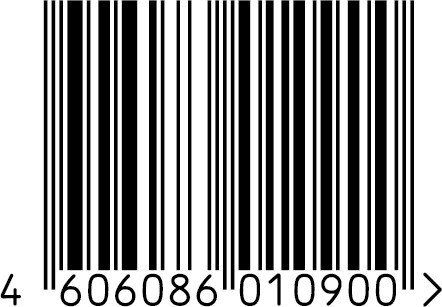
Рис. 2.28 Данные – программа на языке программирования

Итак, информация, как результат *кодирования*, без знания способа соответствующего *декодирования,* оказывается практически бесполезной. Обычно исходный блок информации содержит *данные* о способе декодирования/интерпретации, *данные* о параметрах и настройках способа декодирования и т.д. Такие *данные* называются *метаданными* (или служебными данными). *Метаданные* содержат все необходимые исходные данные для успешной интерпретации. Собственно *декодируемая/интерпретируемая* информация называется *полезными данными* (полезной нагрузкой), а *метаданными* называют *информацию*, которые нельзя отнести к полезной нагрузке.

Внимательный читатель опять увидит рекурсию: метаданные нужно закодировать, следовательно, метаданные должны содержать мета-метаданные для собственного декодирования. В целом это верно. Но, конечно, процесс не продолжается до бесконечности. На практике, например, уже расширение файла говорит о том, какие данные в нем содержатся. Обычно блок метаданных находится, либо в самом начале, либо в самом конце информационного блока и называется *заголовком* (*header*), либо *концевиком* (*trailer*) соответственно. Иногда *метаданные* хранятся и в *заголовке*, и в *концевике*, если это удобно. Бывает, что метаданные сильно «перемешаны» с данными, но их всегда можно легко разделить. Значение в блоке метаданных, по которому восстанавливается *алгоритм декодирования*, обычно называют *сигнатурой* (signature) или подписью алгоритма.

### Кодирование идентификационных данных на примере EAN-13

Мы рассмотрим кодирование *данных* на примере штрих-кода. EAN (European Article Number) – европейский код товара. Хотя теперь расшифровывается как International Article Number, то есть код стал международным. Разберем задачу на примере. Допустим, что имеется штрих-код:



Естественно сканер, прочертив узким световым (лазерным) лучом по символике получит отраженный световой сигнал, который будет слабым в те моменты, когда луч проходил по темному штриху *ч* (поглощаясь темной поверхностью), и, наоборот, сильным, когда луч проходил по светлому штриху *б* (отражаясь светлой поверхностью):



Крайние три полоски (*ч-б-ч*) с обеих и пять полосок в центре (*б-ч-б-ч-б*) сторон задают базовую единицу толщины линии, соответствующую одному биту. Это служебные штрихи, они предназначены для «настройки» сканера на штрих-код, так как штрих-коды на упаковках могут быть самых разных размеров. Таким образом, штрих-код делится на две половины: левую и правую.



Можно видеть, что штрихи (как черные, так и белые) кратны по толщине одной, двум или трем базовым единицам толщины и штрих соответствует одному, двум или трем битам. Единичные биты представлены темными штрихами, нулевые – светлыми. Итак, приведенный штрих код, если внимательно приглядеться, соответствует следующей группе бит (выделены группы служебных бит и две половины полезной нагрузки – левая и правая):

|  |
| --- |
| **101**  010111101001110101111000110100010010000101  **01010**  111001011001101110010111010011100101110010  **101** |

Далее о EAN-13 известно, что он представляет 13 десятичных цифр. Цифры с первой по двенадцатую (слева направо) кодируются группами по семь бит (это явная избыточность нужна для обеспечения помехоустойчивости, впрочем, есть и другие задачи). Разобьем полезные данные на группы по семь бит:

|  |  |
| --- | --- |
| Группа бит | Комментарий |
| **101** | *Метаданные*. Синхронизация сканера. |
| 0101111 | 1-я цифра |
| 0100111 | 2-я цифра |
| 0101111 | 3-я цифра |
| 0001101 | 4-я цифра |
| 0001001 | 5-я цифра |
| 0000101 | 7-я цифра |
| **01010** | *Метаданные*. Синхронизация сканера |
| 1110010 | 7-я цифра |
| 1100110 | 8-я цифра |
| 1110010 | 9-я цифра |
| 1110100 | 10-я цифра |
| 1110010 | 11-я цифра |
| 1110010 | 12-я цифра |
| **101** | *Метаданные*. Синхронизация сканера |

Каждая цифра может быть закодирована одним из трех видов кодов: L-код, G-код, R-код. Схема кодирования следующая:

|  |  |  |  |
| --- | --- | --- | --- |
| Цифра кода с 0 по 11-ю | L-код цифры | G-код цифры | R-код цифры |
| 0 | 0001101 | 0100111 | 1110010 |
| 1 | 0011001 | 0110011 | 1100110 |
| 2 | 0010011 | 0011011 | 1101100 |
| 3 | 0111101 | 0100001 | 1000010 |
| 4 | 0100011 | 0011101 | 1011100 |
| 5 | 0110001 | 0111001 | 1001110 |
| 6 | 0101111 | 0000101 | 1010000 |
| 7 | 0111011 | 0010001 | 1000100 |
| 8 | 0110111 | 0001001 | 1001000 |
| 9 | 0001011 | 0010111 | 1110100 |

Обратите внимание, что R-код представляет собой «инверсию» L-кода, а G-код это R код, записанный в обратном порядке. Декодируем:

|  |  |  |
| --- | --- | --- |
| Группа бит | Цифра | Комментарий |
| **101** |  |  |
| 0101111 | 6 | L-код |
| 0100111 | 0 | G-код |
| 0101111 | 6 | L-код |
| 0001101 | 0 | L-код |
| 0001001 | 8 | G-код |
| 0000101 | 6 | G-код |
| **01010** |  |  |
| 1110010 | 0 | R-код |
| 1100110 | 1 | R-код |
| 1110010 | 0 | R-код |
| 1110100 | 9 | R-код |
| 1110010 | 0 | R-код |
| 1110010 | 0 | R-код |
| **101** |  |  |

Видно, что получаются те же цифры, которые записаны внизу штрих-кода (только для того, чтобы кассир мог ввести их вручную). Не хватает только тринадцатой (нулевой по порядку записи) цифры (написанной на коде крайней справа).

Итак, обращаем внимание на то, что правая (**R**ight) половина цифр в штрих-код е закодирована с помощью **R**-кода, а цифры левой (**L**eft) половины закодированы либо **L**, либо G кодами. Все дело в том, что первыми, кто начал активно использовать штрих-коды с целью идентификации товаров были американцы. Ими был создан код UPC-12 (Universal Product Code), с помощью которого идентифицировались американские товары. UPC-12 кодировал 12 цифр. Европейцы чуть позже решили создать собственный аналог, но совместимый с американским. Они добавили еще одну, тринадцатую, цифру, не изменив ничего в структуре UPC! Как? UPC кодировал левую половину из шести цифр **L**-кодом, а правую – **R**-кодом. Европейские инженеры ввели еще и G-код. То есть одну и ту же цифру в левой половине кода можно закодировать либо L, либо G кодом (при этом сохраняется чередование черных и белых полос: код каждой цифры в левой половине начинается с белого штриха). Дополнительная (нулевая по порядку) цифра кодируется комбинацией L/G кодов, выбранных для кодирования каждой цифры в левой половине штрих-кода:

|  |  |  |  |
| --- | --- | --- | --- |
| 0-я цифра | Комбинация L/G кодов в левой части | 0-я цифра | Комбинация L/G кодов в левой части |
| 0 | LLLLLL | 5 | LGGLLG |
| 1 | LLGLGG | 6 | LGGGLL |
| 2 | LLGGLG | 7 | LGLGLG |
| 3 | LLGGGL | 8 | LGLGGL |
| 4 | LGLLGG | 9 | LGGLGL |

В нашем примере использовалась комбинация LGLLGG, что соответствует цифре 4. Нулевая цифра EAN в данном примере есть 4. Обратите внимание на то, что дополнительная цифра, равная нулю даст вариант EAN, который успешно распознается американским UPC сканером! Совместимость двух технологий достигнута.

Итак, в приведенном примере EAN-13 закодированы цифры: 4606086010900. Мы выполнили переход от штрихов к цепочке из 95 бит (информации). Далее мы выделили в информации определенные части, являющиеся *метаданными* (синхронизация) и *полезной нагрузкой*. Полезная нагрузка была *интерпретирована* как набор из 13 цифр. Казалось бы, мы выполнили переход от *информации* к *данным*. Да, выполнили. Первый. Пора подняться на второй уровень *данных*. Набор цифр также имеет внутреннюю структуру. И в случае идентификации товара она такова:

* Первые три цифры [0,2] это так называемый GS1 префикс, определяющий код страны в которой зарегистрирован производитель. Этот код не определяет страну, где произведен товар, но в большинстве случаев это действительно так. GS1, равные 978 или 979 определяет, что в штрих коде закодирован код ISBN (International Standard Book Number) – международный стандартный номер книги. Смотрите на обложках книг и журналов. ISBN имеет собственную структуру и в нем представлены: код языковой группы (на каком языке книга), код издателя и порядковый номер книги, присвоенный издателем. Поля имеют переменную длину. Код языковой группы следует сразу за GS1, и по штрих коду на книге можно понять на каком она языке. Вот несколько популярных значений кода языковой группы: 0-1 (Английский), 2 (Французский), 3 (Немецкий), 4 (Японский), 5 (Русский).
* Далее следует код компании производителя, имеющий длину от трех до восьми цифр.
* Далее следует код товара в данной компании, имеющий длину от одной до шести цифр.
* Последняя цифра (12-я) это контрольный код, предназначенный для контроля целостности.

GS1 коды некоторых стран: 000-009 (США и Канада), 460-469 (Россия и страны СНГ), 481 (Белоруссия), 482 (Украина), 690 (Китай), 489 (Гонконг), 742 (Гандурас). Кстати, исходя из сказанного об UPC, читателю понятно, почему страны США и Канада имеют код, начинающийся с нулевой цифры?

В частности из приведенного штрих кода можно при желании узнать, что он идентифицирует блокнот для записей фирмы ООО «КАНЦ-ЭКСМО», зарегистрированной в России.

Вот, например, штрих-код на книге «КОД» Чарльза Петцольда:



Язык русский, издательство 7502 («Русская Редакция»), книга 0159 («КОД»).

Можно бы на этом и закончить, но стоит рассказать и о защите целостности, раз уж она есть: последняя цифра штрих кода есть контрольная сумма. Вычисляется она следующим образом:

Цифры, стоящие на четных позициях умножаются на 1, а цифры на нечетных умножаются на 3, и находится сумма результатов. Учтите, что расчеты ведутся по 12 цифрам. Контрольная цифра - это та цифра, которую нужно прибавить к результату, чтобы он делился на 10 без остатка. Напомним, что дополнительная цифра EAN стоит на четной позиции 0, а за ней следуют остальные цифры. Для приведенного примера ISBN = рассчитаем сумму «взвешенных» единицами и тройками цифр:

Очевидно, что контрольная цифра, которую нужно добавить к 116, чтобы результат делился на 10 это 4. Сканеры, считывая штрих-код, проверяют на равенство декодированную цифру контрольного кода и рассчитанную, считывание считается успешным, если они совпадают.

Итак, мы выполнили еще один переход, убедившись, что набор цифр, являясь *данными*, относительно декодированной цепочки бит, в свою очередь является носителем *данных* о товаре. Контрольная же сумма является *метаданными* уже на втором уровне.

**Задания**

1. Декодируйте и проверьте контрольный код:C:\work\projects\IT\Князьков_методичка\2_10\img\barcodes\cut\4601201015112.jpg
2. Декодируйте и проверьте контрольный код:C:\work\projects\IT\Князьков_методичка\2_10\img\barcodes\cut\4605523000702.jpg
3. Декодируйте и проверьте контрольный код:C:\work\projects\IT\Князьков_методичка\2_10\img\barcodes\cut\4606599000962.jpg
4. Декодируйте и проверьте контрольный код:C:\work\projects\IT\Князьков_методичка\2_10\img\barcodes\cut\4605523000931.jpg
5. Декодируйте и проверьте контрольный код:C:\work\projects\IT\Князьков_методичка\2_10\img\barcodes\cut\5997001381649.jpg
6. Декодируйте и проверьте контрольный код:C:\work\projects\IT\Князьков_методичка\2_10\img\barcodes\cut\9785170263271.jpg

### Текстовый и бинарный форматы данных

Среди всех прочих *форматов данных* следует в первую очередь выделить *текстовый* формат, который, как явствует из названия, предназначен в первую для представления текста. Файл, содержащий данные в текстовом формате, называется *текстовым*. Программа, предназначенная для работы с текстовыми файлами, называется *текстовым редактором* (см. Рис. 2.29) и присутствует в базовой поставке любой операционной системы. Текст, как форма представления речи, является важнейшим средством общения людей. Кодирование текста мы подробно обсудили в параграфе 2.5. Еще раз выделим основное: при работе с данными в *текстовом формате* неделимой (атомарной) единицей является *символ* текста (и, как уже известно, ряд управляющих *символов*).

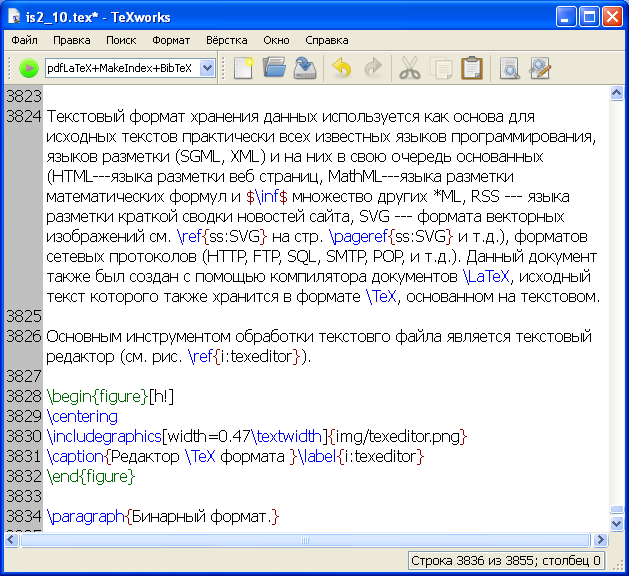


Рис. 2.29 Текстовый редактор

*Бинарный* формат отличается от *текстового* тем, что неделимой единицей при работе с ним является *кодовый символ* (в случае двоичного кодирования – бит). За счет этого бинарный формат во многом компактнее текстового. Например, в текстовом формате для хранения числа 65535 потребуется как минимум пять байт для представления в одной из кодировок текстовых символов «6», «5», «5», «3», «5», а в бинарном формате для хранения такого числа можно отвести всего два байта.

Бинарный формат используют многие прикладные программы для сохранения результатов своей работы в файлах. В бинарном формате сохраняют данные программы-архиваторы, редакторы растровых и векторных изображений, текстовые процессоры, компиляторы, программы для обработки аудио и видео. Форматы многих бинарных файлов стандартизованы (см. например, форматы графических изображений в 2.7.4) и открыты для сторонних разработчиков.

Обычно нет нужды вносить изменения в бинарный файл непосредственно (вручную). Все изменения в бинарный файл вносятся опосредованно, через пользовательский интерфейс соответствующей прикладной программы. Но ежели такая необходимость возникает (например, в исследовательских целях), то основным инструментом является шестнадцатеричный редактор (hex editor), позволяющий вносить изменения на уровне отдельных бит или байт (см. Рис. 2.30).

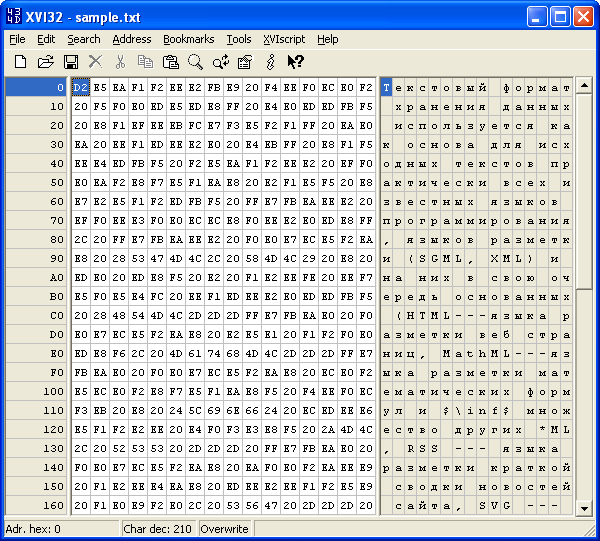


Рис. 2.30 Шестнадцатеричный редактор – низкоуровневое средство для работы с бинарными данными

Файл представляется как массив байт, представленных шестнадцатеричными числами. Двоичные данные файла обычно *интерпретируются* как целые или вещественные числа, битовые маски, фрагменты текстовых данных и т.д. и пользовательский интерфейс hex-редактора позволяет выполнять эту интерпретацию. Практически все hex-редакторы позволяют интерпретировать набор байт как целое число произвольной разрядности, как вещественное число в стандарте IEEE, как строку, позволяют работать как с big-endian так и с little-endian представлениями чисел. Конечно, текстовый файл также можно рассматривать как бинарный и открыть его в шестнадцатеричном редакторе (как и сделано на Рис. 2.30). В силу того, что рядовой пользователь добровольно использовать hex-редактор не будет, в базовую поставку операционных систем этот редактор обычно не входит.

### Язык разметки XML

XML (e**X**tensible **M**arkup **L**anguage) – расширяемый язык разметки, позволяющий описывать *структуру* данных в текстовом формате.

Начнем с простого примера. Допустим, у нас есть текст: «610037, Россия, г. Киров, ул. Ульяновская 22/1, кв. 30». То, что это адрес понятно только человеку. Адрес имеет определенную *структуру*, и человек разберется, в каком бы порядке ни были перечислены отдельные части адреса. Для машинной обработки желательно более четко *структурировать* данные, так как отдельные части этих данных могут понадобиться сами по себе. Можно внедрить структурирующую информацию (метаданные) прямо в текст. Что и делает XML, решая задачу с адресом, например, так: «<адрес> <индекс> 610037 </индекс> <страна> Россия </страна> <город> Киров </город> <улица> Ульяновская </улица> <дом> <номер> 22 </номер> <корпус>1</корпус> </дом> <квартира> 30 </квартира> </адрес>». Видно, где начинается («<адрес>»), а где заканчивается («</адрес>») адрес, индекс, номер дома и т.д. Более наглядно:

|  |
| --- |
| <адрес>  <индекс>**610037**</индекс>  <страна>**Россия**</страна>  <город>**Киров**</город>  <улица>**Ульяновская**</улица>  <дом>  <номер>**22**</номер>  <корпус>**1**</корпус>  </дом>  <квартира>**30**</квартира>  </адрес> |

Теперь поговорим более подробно о *выражениях* языка XML. XML данные сохраняются обычно в файле с расширением \*.xml. В самом начале XML файла содержится выражение следующего вида:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> |

Видно, что указываются используемая версия языка и кодировка, в которой представлен последующий текст. Данное выражение всегда представлено в ASCII. Его можно и не указывать, если вы уверены, что проблем с кодировками и версиями не будет. Далее следует *один* (корневой) *элемент*. *Элемент* (ключевое понятие) имеет следующую структуру.

|  |
| --- |
| <elementName attribute1Name="a1Value" attribute2Name="a2Value">  elementValue  <!--  Тут могут находиться, как значения элемента  атомарных типов: числа, строки и др. типы,  так и вложенные элементы (один или несколько).  Ну и, конечно, КОММЕНТАРИИ!  -->  </elementName> |

Итак, *значение элемента* обычно находится между двумя *тегами*: открывающим и закрывающим. *Тегом* называются данные, заключенные между двумя символами: угловой открывающей «<» и закрывающей «>» скобок. Имя элемента elementName может выбираться совершенно произвольно в зависимости от структуры размечаемых данных (главное, чтобы оно не различалось в открывающем и закрывающем тегах). Итак, открывающий тег в нашем примере:

|  |
| --- |
| <elementName attribute1Name="a1Value" attribute2Name="a2Value"> |

а закрывающий:

|  |
| --- |
| </elementName> |

Перед *именем элемента* в закрывающем теге ставится слеш: «/».

Открывающий тег может содержать произвольное количество разноименных *атрибутов*. *Атрибут* – это пара имя-значение, *имя атрибута* и *значение атрибута* разделены символом равенства «=»:

|  |
| --- |
| attribute1Name="a1Value" |

*Значение атрибута* обязательно записывается в одинарных или двойных кавычках.

Значение элемента elementValue может содержать, либо значение одного из примитивных типов, либо другие элементы в произвольном количестве. Во втором случае, среди нескольких *дочерних* элементов могут быть одноименные. По отношению к дочерним элементам, элемент их содержащий, называется *родительским*. В XML важен порядок следования элементов. То есть, когда вы с помощью программного обеспечения получаете массив *дочерних* элементов, то они пойдут в том же порядке, что и в файле.

Между открывающим и закрывающим тегами элемента можгут находится один или несколько комментариев, заключенных между «<!--» и «-->». Комментарии не могут быть вложенными.

Если значение элемента пусто, то есть все, что нужно знать об элементе, вы задали в *атрибутах*, то можно применить сокращенную запись:

|  |
| --- |
| <elementName attribute1Name="a1Value"/> |

Вместо

|  |
| --- |
| <elementName attribute1Name="a1Value"></elementName> |

Вы, наверное, уже нахмурились и думаете как же, например, в значении атрибута указать двойные или одинарные кавычки, как в значении элемента указать закрывающий тег этого элемента? Да, действительно это проблема, которая решается разделением значений на CDATA и PCDATA. Значения элементов и атрибутов являются PCDATA. PCDATA – Parsed Character Data. То есть *интерпретируемые* символьные данные. Некоторые символы исходного значения должны быть закодированы специальными символами, так как их вхождения запрещены. Например, предопределены: &amp; («&»), &lt; («<»), &gt; («>»), &apos; («'») и &quot; («"»). Таким образом, задача решается:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <book title="Всё о &quot;кавычках&quot; в XML">  Все о закрывающих тегах: &lt;/book&gt; в значении  элемента нужно записать как &amp;lt;/book&amp;gt;.  Символ Unicode (например &#9763;) можно вставить,  указав его десятичный код так &amp;#9763;  </book> |

Если набрать этот текст в текстовом редакторе и сохранить в файле с расширением XML, а затем открыть с помощью любого web-браузера, то увидим примерно следующее:

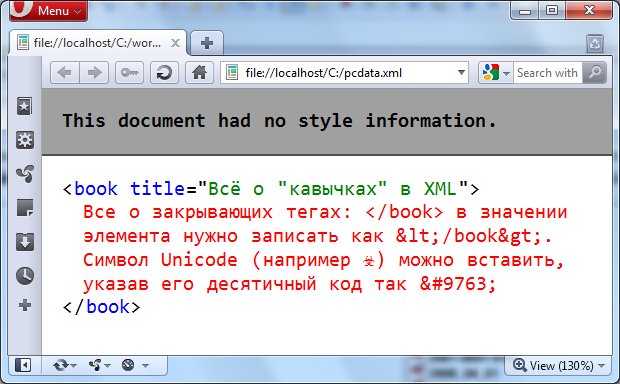


Рис. 2.31 PCDATA в XML

Также можно задавать десятичные коды символов Unicode, например символ с кодом 9763 можно вставить в PCDATA так: «&#9763;».

Часто возникает необходимость хранить внутри XML документов исходные тексты программ, так как многие форматы, производные от XML (например, формат векторной графики SVG), являются интерактивными и в процессе работы с документами таких форматов могут выполняться содержащиеся в этих документах программы – скрипты. Чтобы не заменять вхождения спецсимволов в текстах программ, часто применяют значения CDATA. CDATA – Character Data. То есть *не интерпретируемые* символьные данные, которые располагаются между вхождениями «<![CDATA[» и «]]>». Конечно, значениями CDATA могут быть не только тексты программ, но и любые другие данные, в интерпретации которых нет необходимости. Сравните значения элементов sample в следующем примере:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <scripts>  <!-- каторжная работа по устранению вхождений.  Удобочитаемость программы = 0 -->  <sample type="PCDATA">  // комментарий к тексту программы  if (this-&gt;x &lt; 5 &amp;&amp; values[0] != 1) {  alert(&quot;Сообщение&quot;);  }  </sample>  <!-- используя CDATA пишем код как есть -->  <sample type="CDATA">  <![CDATA[  // комментарий к тексту программы  if (this->x < 5 && values[0] != 1) {  alert("Сообщение");  }  ]]>  </sample>  </scripts> |

Очевидно, раздел CDATA не может содержать вхождений «]]>». А в программе вполне может встретится что-нибудь наподобие:

|  |
| --- |
| <![CDATA[  //следующая строка закроет CDATA :(  if (a[b[0]]>3) { //катастрофа!  //что-нибудь делаем... увы, это самообман.  //Целостность XML нарушена  }  ]]> |

Очевидно, что на третьей строке счастливая жизнь и закончится… Впрочем, внутри границы CDATA «]]>» пробелов быть не может, а вот всем без исключения интерпретаторам скриптов все равно сколько идет пробелов между термами. Можно приучить себя ставить хоть один пробел перед знаком больше[[31]](#footnote-31)?

В XML еще много важных понятий, которые мы рассматривать не будем, остановившись на этом уровне, которого нам хватит для понимания остальных тем. Но совершенно нелишними для читателя будут знания о схемах и о пространствах имен XML. Язык XML универсален, определив специфичные имена элементов и атрибутов (а также смысловую нагрузку на них), специалисты в разных областях получили множество специализированных языков: MathML (разметка математических формул), HTML[[32]](#footnote-32), SVG(векторная графика, см. далее), RDF (и производный от него, RSS для описания новостных лент сайтов), BioML (применяется в биологии), и множество других…

**Практические задания**

1. Опишите с помощью XML данные (не менее 10 важнейших разноименных элементов):

|  |  |
| --- | --- |
| Вариант | Данные |
| 1 | Конфигурация компьютера |
| 2 | Конфигурация телевизора |
| 3 | Автомобиля |
| 4 | Библиотечного формуляра |
| 5 | Расходного кассового ордера |
| 6 | Счет на услуги ЖКХ |
| 7 | Квитанция об оплате штрафа |
| 8 | Схема кодирования на |
| 9 | Промышленное предприятие |

1. Задайте значение в PCDATA.

|  |  |
| --- | --- |
| Вариант | Значение PCDATA |
| 1 | <![CDATA[ "CDATA" ]]> |
| 2 | <!-- <data>Это не комментарий</data> --> |
| 3 | <noelement noattribute="value 'value' is not value "/> |
| 4 | " < &lt; &quot; &amp; &quot; &gt; > " |
| 5 | If (символ == '☠') alert("В ящике пираты!"); |
| 6 | Пустыня. ☀. Вода ☞5 км; Вода ☞ 2 км; Вода ☟1 км; >:( |
| 7 | <А/> & <Б/> сидели на ♖. &amp; остался на трубе |

### Язык логической разметки печатных документов LaTeX

Конечно, XML универсален и позволяет при желании выполнить разметку текстовых данных для *любых* задач. Но для непосредственного редактирования в текстовом редакторе XML не очень удобен. Настольная издательская система LaTeX определяет специализированный (а следовательно и более удобный) язык для логической разметки текста[[33]](#footnote-33).

Задача красивого оформления книгопечатной продукции возникла вместе с книгопечатной продукцией. От того, как *нарисована* буква на бумаге или экране зависит очень много! Создать хороший *шрифт* – задача для *художника*, владеющего знаниями о гармоничном восприятии. Часть этих знаний оформлена в математических формулах и законах. Написать берущее за душу произведение – задача *писателя*. И писатель не должен заботится о том как лучше нарисовать букву «А»: с засечками на ножках или без, и как в этом случае она будет смотреться рядом с буквой Б? На заре компьютерной техники эти задачи решались заново[[34]](#footnote-34).

*Писатель* *не должен* думать об оформлении! Он должен *логически* выстроить свое произведение: например, выделить части, главы, разделы, параграфы, введение, написать аннотации, формулы, примечания, эпиграфы и т.д. Выбрать шрифты, визуально выделить названия частей и глав, определить подходящие отступы, выделить буквицу, оформить узором поля, оформить надлежащим образом аннотации, сформировать оглавление, вообще – *оформить* (сверстать) печатное издание так, как это требуется правилами, должен *издатель*.

В настоящее время WYSWYG[[35]](#footnote-35) редакторы (Open Office writer, Microsoft word) совмещают процесс создания с процессом верстки. Порой это приносит неудобства: редактировать текст в том виде, в котором он будет *отображаться* иногда крайне неудобно. Более того, на выходе WYSWYG редакторов, как правило, получается один бинарный файл (пусть теперь это целая заархивированная директория, но сути дела это не меняет). Бинарный – не самый удобный формат для коллективной работы над изданием или для отслеживания истории его развития. LaTeX же в полном объеме позволяет использовать все преимущества систем контроля версий и систем коллективной разработки, без которых программисты не мыслят жизни своей.

В LaTeX документ может создаваться в простейшем текстовом редакторе. Определенные *выражения* языка разметки влияют на отображение фрагментов текста. Исходный текст поступает на вход программы-преобразователя форматов и на выходе получается файл в формате, удобном для печати (чаще это бинарный формат pdf[[36]](#footnote-36)). При этом существует огромное количество шаблонов оформления документа. Так для того, чтобы оформить текст, например по книжным правилам или по правилам оформления научных статей, достаточно изменить всего одну строку исходного текста и вновь выполнить преобразование. Мы рассмотрим простейший пример документа LaTeX. Он несложен, и для его понимания отметим лишь то, что комментарии в исходном тексте документа, служащие для пояснений, начинаются с символа % и заканчиваются в конце строки. Итак, исходный текст документа:

|  |
| --- |
| %Это комментарий  %вначале выполняется общая настройка  %указывается класс документа (влияет на отображение). В CTAN архивах  %находится огромное множество классов для оформления статей, книг,  %буклетов, презентаций и сборников...  \documentclass{article} %делаем статью. Хотите книгу?  %Поставьте book вместо article...  \usepackage[cp1251]{inputenc} %задаем кодировку исходного текста  \usepackage[russian]{babel} %включаем поддержку русского языка. В частности  %корректные переносы слов на новую строку  %общие данные для документа  \title{Кодирование печатных документов в \LaTeX} %название  \author{М.~М.~Шихов} %автор  \date{15 декабря 2011 года} %дата. вставка текущей даты: \date{\today}  \begin{document} %начало тела документа  \maketitle %печатает данные титульного листа  %как именно определяет команда \documentclass см. использование выше  \begin{abstract} %начало аннотации  Это аннотация. Простой пример исходного текста, сравнивая  который с результатом, легко освоить основы \LaTeX.  \end{abstract} %конец аннотации  %одной строкой формируем содержание  \tableofcontents  \section{Текст заголовка раздела} % помечаем заголовок раздела  %подразделы формируются командами \subsection \subsubsection  Текст раздела. Красивое оформление исходного текста  облегчает работу с ним. Исходные тексты больших документов  можно разнести по нескольким файлам.  \subsection{Текст заголовка подраздела}  \LaTeX имеет замечательные возможности описания текстом математических формул.  %одна или несколько пустых строк обозначают конец абзаца  Вот новый абзац. Вы можете записать формулу прямо в тексте абзаца $a^2=b^2+c^2$.  Вы можете записать формулу отдельной строкой \[a=\sqrt{ b^2+c^2}.\]  Можете пронумеровать её  \begin{equation}  \label{eq:pifagor} %определить понятное имя для метки,  a=\sqrt[2]{ b^2+c^2} %чтобы затем сослаться на эту формулу  \end{equation}  И затем в любом месте документа сослаться на формулу \ref{eq:pifagor}.  \end{document} %конец документа |

Результат преобразования будет выглядеть так:

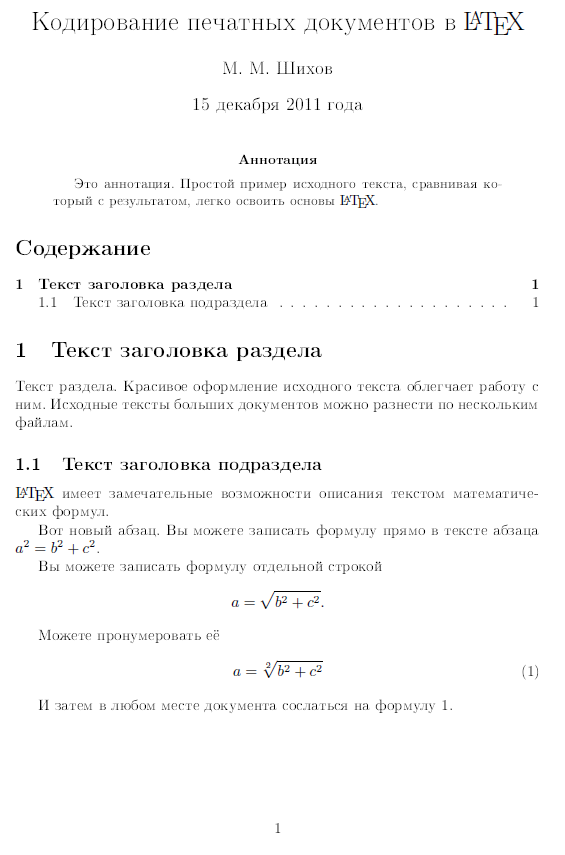


Рис. 2.32 Результат преобразования исходного текста LaTeX

Еще одно достоинство языка LaTeX – поддержка математических формул. В первую очередь это средство создавалось для красивого оформления математических текстов. Приведем несколько примеров.

|  |  |
| --- | --- |
| Исходный текст формулы в формате LaTeX | Результат преобразования |
| a^X |  |
| a\_i |  |
| x=\frac{1}{1+\frac{1}{1+frac{1}{x}}} |  |
| b\_j^X=\sum\_{i=0}^{2}a\_{3\cdot j+i}^X\cdot 2^i |  |
| Y=\sum\_{j=1}^{\infty}\frac{a\_{-j}^Y}{m^{j}} |  |

В последнее время LaTeX уступает в популярности WYSWIG редакторам, которые много проще в освоении. Но остается и продолжает развиваться огромное наследие в виде CTAN[[37]](#footnote-37) архивов, где собраны коллекции самых специфичных стилей оформления: с помощью LaTeX можно создавать математические тексты[[38]](#footnote-38), технические документы, оформленные по всем правилам (например, ЕСКД), ноты, форматировать код, описывать электронные схемы, и т.д.

**Практические задания**

1. Восстановите исходный текст формулы в LaTeX:

|  |  |
| --- | --- |
| Вариант | Формула |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |

1. В какие формулы скомпилируется исходный текст LaTeX:

|  |  |
| --- | --- |
| Вариант | Исходный текст LaTeX |
| 1 | $ a^{\frac{1}{c}} =\sqrt[c]{a} $ |
| 2 | \[ a^{b+c}=a^{b}a^{c} \] |
| 3 | \[ \sqrt{a}=a^{\frac{1}{2}} \] |
| 4 | Для возведения в степень  справедливо равенство \ref{eq:power}  \begin{equation}  \label{eq:power}  (a^b)^c=a^{bc}  \end{equation} |
| 5 | \[ m\sum\_{i=0}^{n}a\_i=\sum\_{i=0}^{n}ma\_i \] |
| 6 | \[ \frac{m}{\sum\_{i=0}^{n}a\_i}=\frac{1}{\sum\_{i=0}^{n}\frac{a\_i}{m}}; m\neq 0 \] |

### Форматы графических данных

Прежде следует сказать несколько слов о том, каким образом *кодируется* цвет. Различают *монохромные* и *полноцветные* изображения.

*Монохромное* изображение содержит цвет одного спектра (длины волны), воспринимаемый человеком как один оттенок. Примерами монохромных изображений могут быть черно-белые фотоснимки, рисунки карандашом, тушью. Такие изображения также называют *полутоновыми*. Для представления оттенков одного цвета достаточно одной компоненты. Как правило, эта компонента кодируется одним -разрядным целым двоичным числом. При этом получается полутоновая шкала и значение 0 шкалы соответствует самому темному оттенку, а самому светлому. Когда в шкале присутствует лишь два оттенка: тёмный и светлый. Нулевому биту шкалы соответствует, например, темный оттенок, а единичному биту – светлый. Иногда битам соответствуют разные *цвета*, и в этом случае изображение называют *бинарным*, а не *монохромным*.

*Полноцветное* изображение. Все богатство красок достигается за счет комбинирования определенных физических составляющих цвета. Способ численного представления цветов называется *цветовой моделью*. Таких способов довольно много, и мы обсудим наиболее употребимые из них.

Прежде скажем несколько слов о восприятии цвета человеком. Сетчатка глаза человека содержит два типа сенсоров, называемых *палочками* и *колбочками*. Палочки отвечают за восприятие *яркости*, а колбочки – за восприятие *цвета*. В одном глазу около 100 миллионов палочек и около 6 миллионов колбочек. Видимо, глаз выполняет «предобработку» данных с сенсоров, потому как в мозг от глаза несут сигнал около одного миллиона нервных волокон. Для того, чтобы колбочки нормально воспринимали цвет, необходимо хотя бы несколько . Монитор обычно дает около для белого цвета и для черного.

По чувствительности к свету определенной длины волны различают три вида *колбочек* (см. Рис. 2.33). Пик чувствительности первых приходится на длины волн около 600 нм ( м} (воспринимаются как оранжевый цвет, а не красный, как некоторые, слышавшие о модели RGB, ожидали), вторых на длины волн около 550 нм (зеленый цвет), третьих на 440 нм (синий/голубой цвет). Следует отметить, что «*видимый*» диапазон волн нахдится в пределах нм. Приведенные кривые чувствительности колбочек являются математической интерпретацией результатов множества экспериментов по исследованию восприятия цвета.

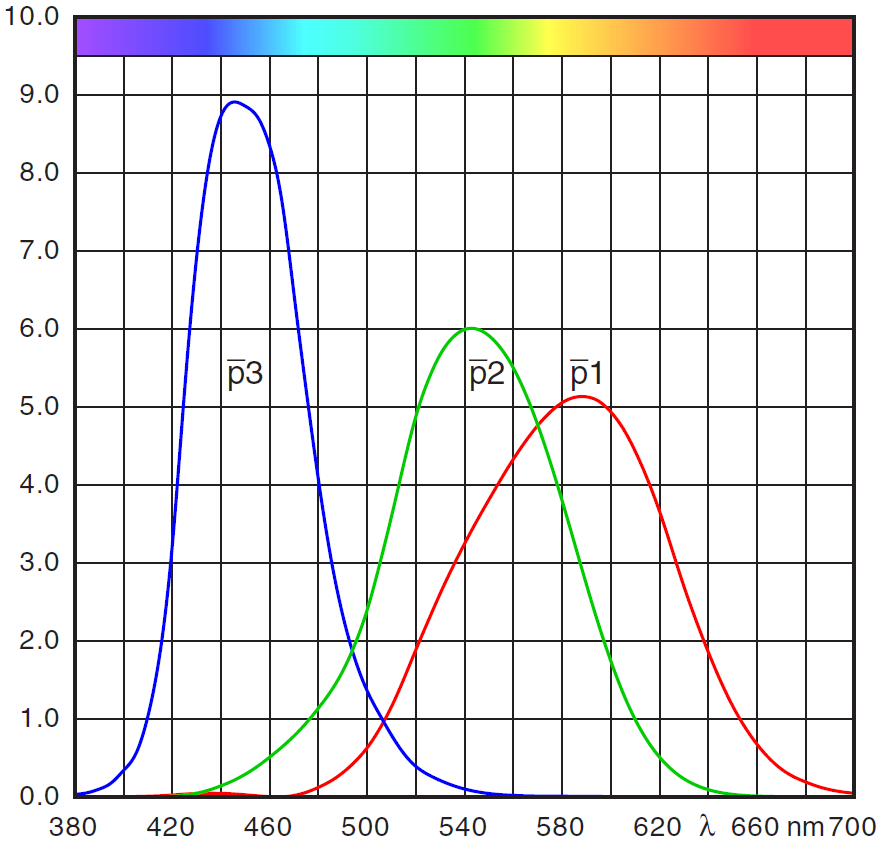


Рис. 2.33 Чувствительность колбочек к волнам определенной длины

Удивительным является то, что мозг «восстанавливает» утерянную часть изображения, попавшую на «слепое пятно» (особая область сетчатки, от которой информация в мозг не поступает). Так что все намного сложнее, конечно.

Обозначим энергетическую спектральную кривую светового потока, падающего на отдельно взятую колбочку. Значение функции соответствует мощностному вкладу волн с длиной волны в общий световой поток. Колбочка -го вида (одна из ) выполняет следующее математическое преобразование:

|  |  |
| --- | --- |
|  | (2.4) |

где – весовая функция чувствительности (см. ) -го вида колбочек.

То есть, если на определенный участок сетчатки падает поток , то от колбочек этого участка в соответствии с формулой (2.4) пойдет в мозг тройка (от каждого вида колбочек) параметров: . Эту тройку чисел будем называть *стимул*. Т.е. участком сетчатки выполняется отображение *спектра* на *стимул*:

Естественно, это отображение не является взаимно однозначным. Одному и тому же *стимулу* соответствует бесконечное множество возможных спектров . При этом *световые потоки* со спектрами, дающими одинаковый *стимул* мозгу, будут восприниматься как один и тот же *цвет*. То есть для получения, какого бы то ни было цвета вовсе не обязательно воспроизводить *в точности* его спектр, достаточно создать спектр, который возбудит сенсоры *сходным* образом. Этот факт лежит в основе конструкции всех полноцветных устройств вывода.

Эксперименты по исследованию восприятия цвета были предприняты ученым Германом Гюнтером Грассманом (Hermann Gunther Grassmann, 1809-1877) около 1853 г. Суть этих экспериментов заключалась в следующем (см. Рис. 2.34). Имеются три лампы, дающие линейно независимые спектры: ; мощностной вклад каждой лампы будет регулироваться соответствующим весовым коэффициентом: и создавать *впечатление* цвета :

|  |  |
| --- | --- |
|  | (2.5) |

имеется лампа, дающая спектр , который создает впечатление цвета . Итак, можно ли подобрать так, что впечатления от цветов и будут совпадать ()?



Рис. 2.34 Эксперимент Грассмана по восприятию цвета

Во многих случаях ответ: да. Например, . В некоторых случаях ответ: нет. Но, если переместить одну из ламп налево (к лампе ), то, оказывается, можно: Несмотря на практическую невозможность «вычитания» части спектра, можно записать .

В модели CIE-RGB только один компонент может быть с отрицательным весом – и этот компонент . Потому RGB устройства, как будет показано ниже, не в состоянии воспроизвести все богатство красок реального мира. В упомянутой модели, соответственно R=700 нм (красный), G =546.1 нм (зеленый) и B=435.8 нм (синий) и мощностной вклад нормирован коэффициентами, соответственно , , (см. Рис. 2.35).

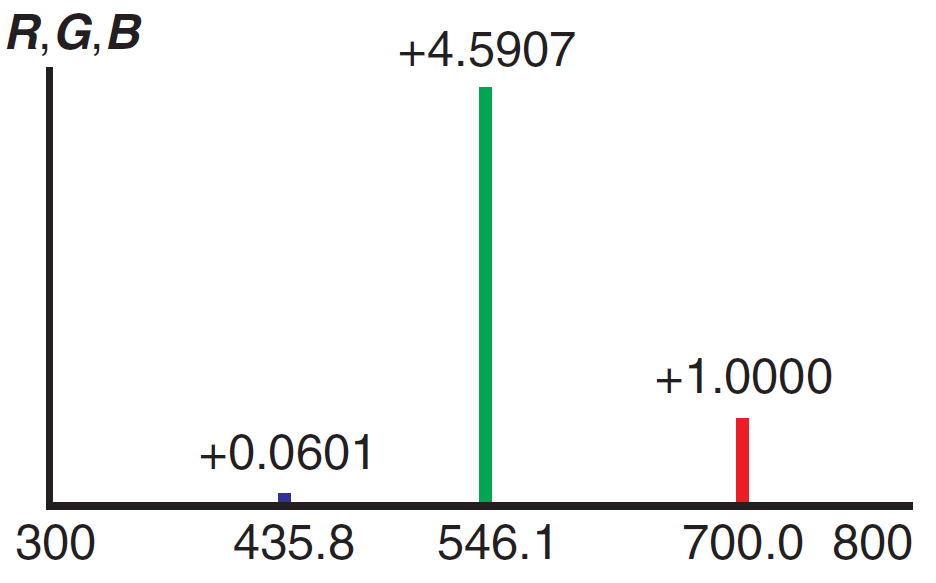


Рис. 2.35 Образующие спектры модели CIE-RGB

Нормализованные весовые функции ,, этой модели представлены на. Видно, например, что для «чистого» спектра, представленного лишь излучением с длиной волны , требуется отрицательное значение веса для компоненты красного цвета.

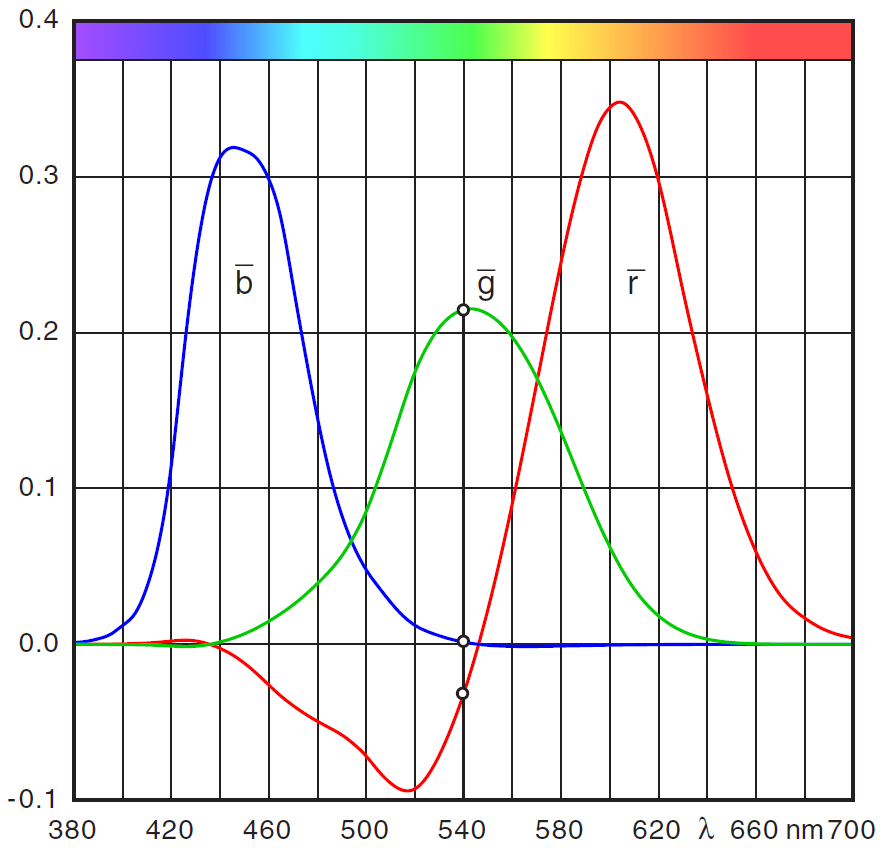


Рис. 2.36 Весовые функции модели CIE-RGB

При этом для произвольного спектра , создающего впечатление цвета , коэффициенты для

дающего то же впечатление (формирующего тот же стимул), могут быть рассчитаны по формулам:

|  |  |
| --- | --- |
|  | (2.6) |

где – нормирующий коэффициент и выбирается исходя из практических нужд.

Итак, с помощью трех чисел, поставленных в соответствие яркости каждой из компонент красного , зеленого и синего цветов можно получить большую часть из всех воспринимаемых человеком цветов. Изображение, разделенное на три RGB компонента, приведено на рисунке Рис. 2.37[[39]](#footnote-39).



Рис. 2.37 Изображение и составляющие его RGB компоненты

Цветовая модель CIE-RGB является практически общепринятой. Применяется во многих растровых устройствах, например, мониторах, телевизорах и проекторах. С целью создать универсальную модель с неотрицательными весовыми функциями, CIE определила *абстрактную* модель[[40]](#footnote-40) CIE-XYZ. Образующие векторы этой модели X,Y,Z содержат отрицательные значения вкладов RGB компонент. То есть каждый «орт» содержит определенные вклады (в том числе и отрицательные) синего, зеленого и красного спектров. Зато разложить любой спектр по таким образующим можно не прибегая к «вычитанию». При этом образующие моделей XYZ и RGB связаны обычным линейным преобразованием:

где , , а – это матрица:

Перейти от модели XYZ к модели RGB можно с помощью обратной матрицы : :

Образующие векторы модели CIE-XYZ можно получить, используя матрицу . Например, вектору соответствует вектор . Зная, что вектор нормирован (см. Рис. 2.35), находим, что орту соответствует . Образующие векторы модели CIE-XYZ приведены на рисунке Рис. 2.38.

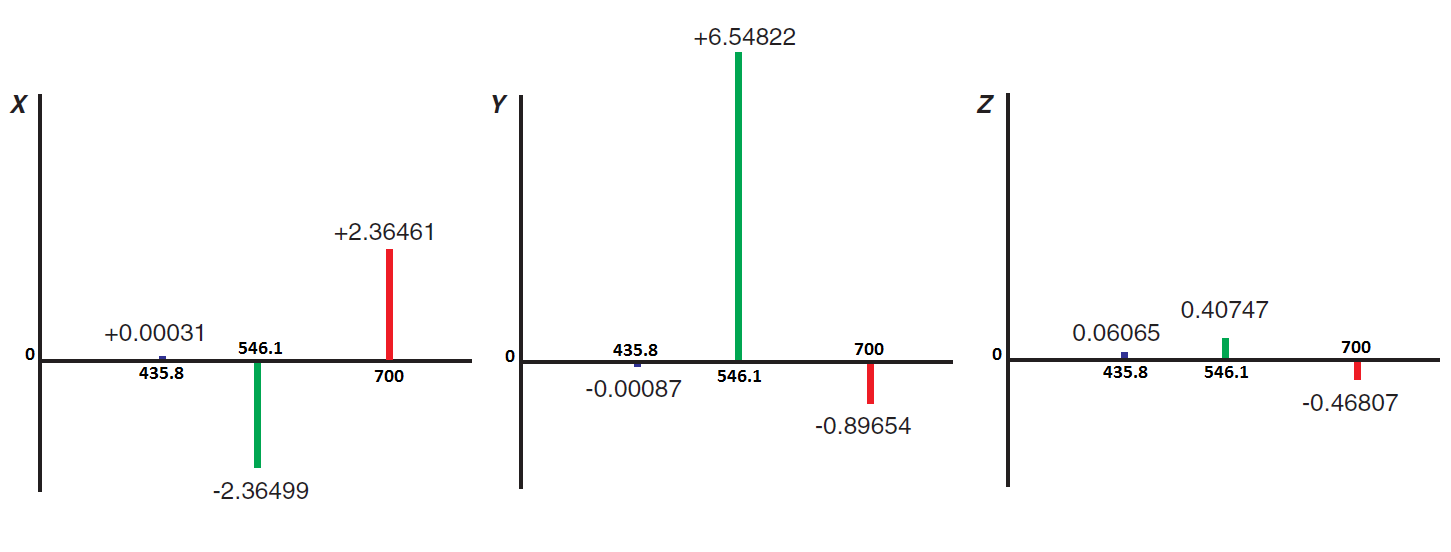


Рис. 2.38 RGB компоненты образующих векторов модели CIE-XYZ

Весовые функции ,, модели XYZ получаются из соответствующих функций ,, модели RGB с помощью матрицы . И, как видно, они неотрицательны на всём множестве определения (см. Рис. 2.39 )

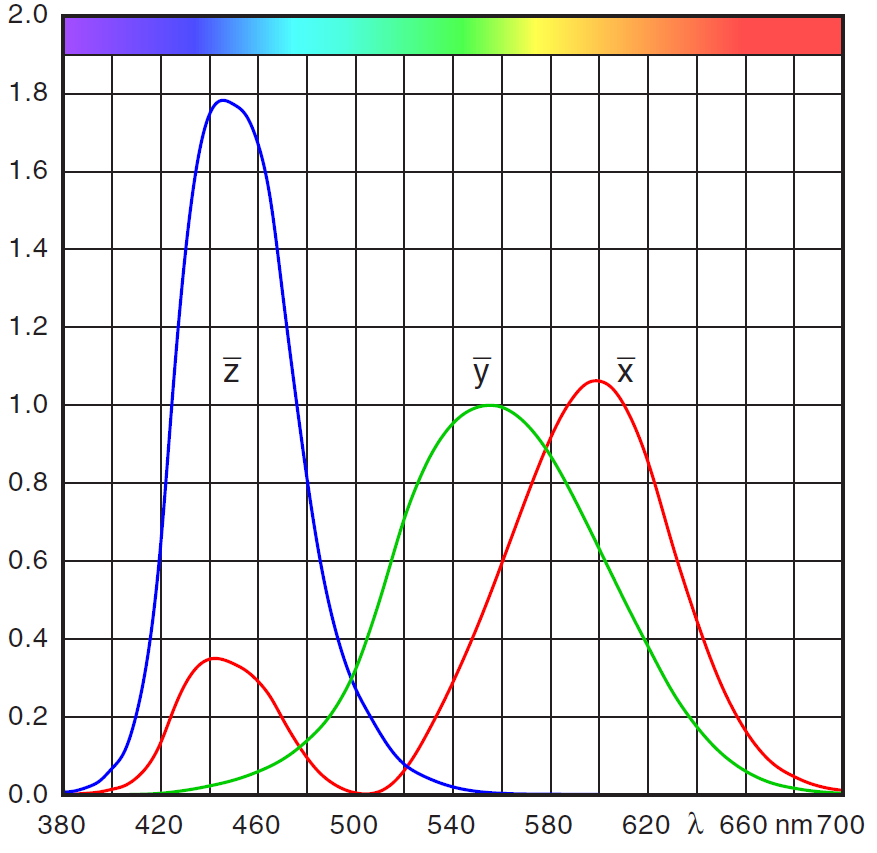


Рис. 2.39 Весовые функции модели CIE-XYZ

Для произвольного спектра , создающего впечатление цвета , коэффициенты для

дающего то же впечатление (формирующего тот же стимул), могут быть рассчитаны по формулам:

|  |  |
| --- | --- |
|  | (2.7) |

На основе универсальной модели CIE-XYZ обычно определяют и все остальные модели, а также хроматическую диаграмму, представляющую все возможное множество воспринимаемых человеком цветов. Хроматические числа , , определяются следующим образом:

|  |  |
| --- | --- |
|  | (2.8) |

При этом, так как хроматические числа неотрицательны, справедливо . Поэтому в качестве хроматической диаграммы рассматривают не саму диаграмму, которая расположена на плоскости , а её проекцию на плоскость (см. Рис. 2.40). В силу того, что по известным , всегда можно восстановить . А по известным , и можно восстановить все остальные параметры цвета: .

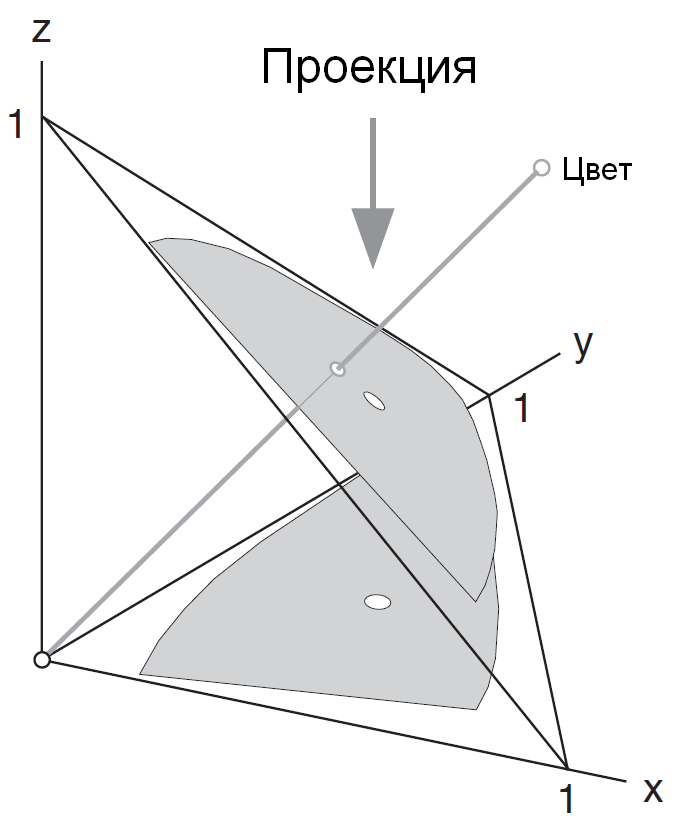


Рис. 2.40 Проекция хроматической диаграммы на

Хроматическая диаграмма получается следующим образом. Для всех длин волн видимого диапазона ([380,700] нм) будем получать значения , по формулам (2.8) исходя из: ,, . Это значит, что мы рассматриваем такие спектры, в которых ненулевой мощностной вклад вносит только свет с длиной волны (монохромат). При этом мы получим кривую на плоскости , каждой точке которой соответствует цвет монохромата. Соединив первую и последнюю точку полученной кривой (этим точкам соответствуют длины волн монохроматов 380 и 700 нм) прямой линией (эта линия называется пурпурной) получим замкнутую фигуру на плоскости. Очерчивают контур этой фигуры цвета монохроматов, а внутри содержатся все возможные цвета, полученные в результате смешения монохроматов (т.е. всех возможных спектров). Хроматическая диаграмма представлена на рисунке Рис. 2.41 Хроматическая диаграмма CIE-XYZ.

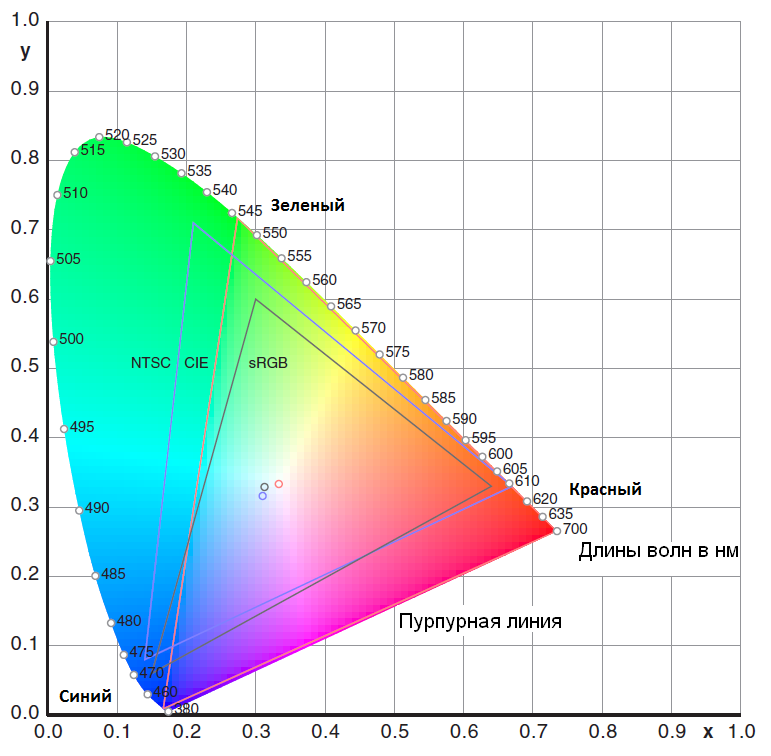


Рис. 2.41 Хроматическая диаграмма CIE-XYZ

Если мы возьмем две точки внутри хроматической диаграммы, которым пусть соответствуют цвета , , то цвета получаемые в результате их смешения будут находиться на отрезке, эти две точки соединяющем. Если взять три точки, то в результате смешения соответствующих им цветов, мы получим цвета, находящиеся внутри треугольника, образованного этими точками. Как видно, какие бы три цвета мы не взяли, треугольник не покроет всей хроматической области. Поэтому, читая этот текст и глядя на хроматическую диаграмму (Рис. 2.41) знайте, что на каком бы устройстве вывода не было бы получено её изображение – многие цвета на ней выглядят не так, как должны… На рисунке также представлены треугольники, соответствующие популярным моделям, применяемым в «полноцветных» технических устройствах: NTSC, CIE(CIE-RGB), sRGB. Хроматическая диаграмма часто используется для указания диапазона отображаемых устройством цветов. Этот диапазон часто называют заимствованным словом *гамут* (англ. gamut – диапазон, область). Хроматическая же диаграмма определяет *гамут* человека.

Мы рассмотрели наиболее популярную в технике модель RGB и абстрактную модель XYZ. Скажем несколько слов и о других моделях, весьма активно используемых на практике.

Цветовая модель CMYK для получения цвета использует смешение четырех цветов: **C**yan (Голубой), **M**agenta (Пурпурный), **Y**ellow (Желтый), Blac**k** (Черный) и давших название этой модели. Эта модель создана для печатающих устройств. Если монитор, формируя цветное изображение, *излучает* свет определенных длин волн, то принтер наносит на бумагу краску, которая будет *поглощать* свет определенных длин волн из солнечного (или близкого к нему) спектра. В отличие от модели RGB, которая относится к *аддитивным*, CMYK – *субтрактивная* модель. Голубой, пурпурный и желтый цвета служат дополнениями для красного, синего и зеленого соответственно. Например: голубая поверхность поглощает красный цвет; если комбинировать желтую и пурпурную краски, поглощающие зеленый и синий, то это приводит к получению красного цвета. Если использовать все три краски – результат будет черным (по крайней мере теоретически). То есть для получения черного цвета придется комбинировать все три краски, что приводит к большому их расходу, поэтому используют дополнительно еще и чисто черную краску. Используют черную краску еще и по той причине, что большинство печатных работ выполняются в черном цвете. Буквы данного текста черные. По крайней мере должны были быть… Обратите внимание на «пробу» красок на полях некоторых цветных газет и вы увидите, что там действительно используется CMYK модель (см.).

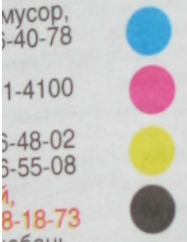


Рис. 2.42 Для печати цветных газет используется модель CMYK

Переход от RGB модели к CMYK достаточно прост: пусть и , , компоненты (фактически под , , тут мы имеем в виду в формуле (2.8)) и , , , изменяются в пределах [0,1], тогда

Обратный переход также прост:

Цветовая модель YCrCb во многом подобна цветовой модели используемой в телевидении. Компонент Y определяет *яркость*, а компоненты Cb и Cr определяют *цветность*. Cb задает *синеву* изображения, а Cr – *красноту*. Раз уж речь зашла о телевидении, то черно-белому телевизору достаточно компоненты Y для воспроизведения монохромного изображения, а цветной телевизор использует все компоненты для получения цветного изображения. Конечно, уже давно в прошлом черно-белые телевизоры, но, тем не менее, они долгое время сосуществовали с цветными. В предположении, что все компоненты изменяются в пределах [0,1], переход от RGB к YCbCr осуществляется так:

Алгоритмы сжатия данных с потерями, например Jpeg используют данную модель, в которой наиболее важная информация сосредочена в одном из компонентов. Это позволяет добиться большего сжатия путем включения большего объема информации по компоненту Y, чем по компонентам Cb, Cr.

Разговор о моделях можно продолжать еще долго и вне обсуждения останутся такие достойные внимания модели, как CIE Lab, HSB, HSV, HLS, и многие, многие другие.

Иногда в исходном изображении можно выделить не так уж много цветов. Тогда можно определить массив, в котором будут находиться описания каждого используемого цвета в выбранной цветовой модели. Тогда в дальнейшем, там, где потребуется указать цвет (например, для определенной точки или линии изображения) будем указывать *индекс* этого цвета в массиве. Такой подход позволяет сэкономить память для представления изображения. Впрочем, были в прошлом причины и весомее: устройства вывода просто не позволяли выводить большое количество цветов, например в модели RGB доступно цветов, а видеокарта позволяла выводить только 256 цветов (произвольно выбранных, а не фиксированных, конечно) из всего множества. Массив, хранящий данные о цветах изображения, называют *палитрой*.

Мы обсудили представления цветов в изображении, но стоит сказать несколько слов и о способах *формирования* самого *изображения* (т.е. о *графике*). Различают *векторную* и *растровую* графику. Представьте себе, что вам нужно сохранить чертеж детали, выполненный инженером и картину, написанную маслом художником. В первом случае приходит мысль хранить информацию о линиях, точках, надписях на чертеже, их размерах и взаимном расположении. То есть выделить достаточно сложные геометрические объекты, из которых можно составить изображение: точки, прямые и кривые линии, многоугольники, дуги и сектора, текст. Во втором случае такой подход неприменим (сложно выделить в несчетном множестве наложенных друг на друга мазков кистью какие-то геометрические примитивы…): проще разбить плоскость картины на множество точек (чем точек больше, тем лучше) и запомнить цвет каждой точки (цветная точка изображения называется *пикселем* (англ. pixel – точка)). Итак, в первом случае мы используем *векторную* графику, а во втором *растровую*. Соответственно различают *векторный* и *растровый* форматы графических данных. Примерами растровых форматов являются: BMP (Bitmap Picture), GIF (Graphics Interchange Format), PNG (Portable Network Graphics), TIFF (Tagged Image File Format), JPEG (Joint Photographic Experts Group, по названию организации-разработчика) и т.д. векторных: SVG (Scalable Vector Graphics ), WMF (Windows MetaFile), VSD (Visio Diagram) и т.д. Собственно и устройства вывода разделяют на векторные и растровые, например, плоттер – векторное устройство, а монитор и принтер – устройства растровые. В настоящее время растровые устройства вывода вытесняют векторные. Сходите в музей.

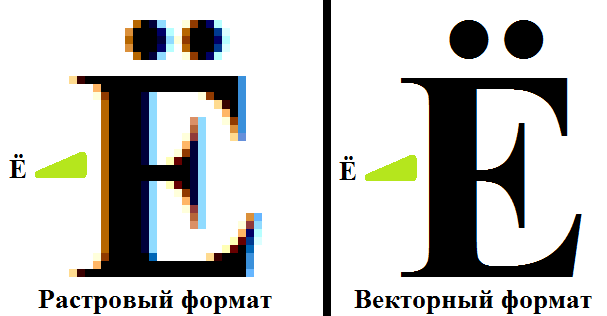


Рис. 2.43 Масштабирование: растровый формат vs векторный формат

Тот и другой формат имеют определенные достоинства и недостатки. Растровый формат плохо масштабируется: если мы увеличим линейные размеры рисунка, то увеличится и размер отдельной точки (ведь их количество останется неизменным) и в определенный момент «мозаичность» изображения станет заметной (см. Рис. 2.43). В случае векторного формата масштабирование никаких проблем не представляет: соответствующим образом масштабируются примитивы, составляющие изображение (кстати, поэтому шрифты (изображения символов текста) хранятся в векторном формате). Подавляющее большинство устройств вывода графической информации – растровые, поэтому векторный формат приходится постоянно преобразовывать в растр. Растровый формат по определению достаточно требователен к ресурсам памяти, поэтому большинство растровых форматов используют алгоритмы *сжатия*. При этом разделяют алгоритмы сжатия с потерями, когда незначительная (а иногда и значительная, но не заметная для сенсорной системы человека) часть данных о цвете теряется, и без потерь, когда по сжатым данным исходное изображение восстанавливается в первозданном виде.

#### Формат BMP

**B**IT**M**A**P** – битовая карта. Бинарный растровый формат графических данных от компании Microsoft. Относится к числу простейших. Использует цветовую модель RGB. Поддерживает изображения с 1, 4, 8, 16, 24, 32 битами на пиксель. Поддерживает простые алгоритмы сжатия Rle4, Rle8 для изображений с 4 и 8 битами на пиксель. Формат прошел несколько стадий совершенствования и в результате имеется несколько поддерживаемых версий. Мы рассмотрим базовую версию формата.

Многобайтовые числа хранятся в формате little-endian, то есть младший байт числа, хранится по младшему адресу. BMP файл имеет структуру, приведенную на рисунке Рис. 2.44.

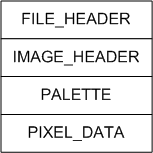


Рис. 2.44 Структура BMP файла

Как видно из рисунка, файл состоит из идущих друг за другом блоков данных: FILE\_HEADER (файловый заголовок – содержит метаинформацию о файле), IMAGE\_HEADER (заголовок изображения – содержит метаинформацию об изображении), PALETTE (палитра. иногда может отсутствовать, например при 24 битах на пиксел), PIXEL\_DATA (данные пикселов, то есть информация о точках изображения). Рассмотрим каждый блок в отдельности.

Блок FILE\_HEADER представлен на рисунке Рис. 2.45. Слева указаны смещения в байтах (в 16-системе счисления) относительно начала блока, в скобках после имени поля указан его размер в байтах.

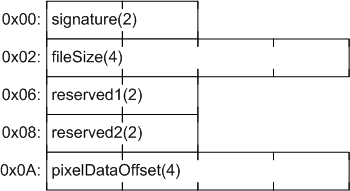


Рис. 2.45 Блок FILE\_HEADER

Назначение полей следующее:

* signature Сигнатура служит для идентификации формата и всегда содержит ASCII-символы «BM» (Hex:0x42,0x4D. BitMap).
* fileSize Размер файла в байтах.
* reserved1 Зарезервировано и пока не используется.
* reserved2 Зарезервировано и пока не используется.
* pixelDataOffset Смещение в байтах относительно начала файла до начала блока PIXEL\_DATA.

Блок IMAGE\_HEADER представлен на рисунке Рис. 2.46.

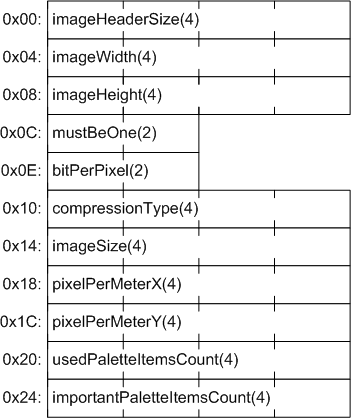


Рис. 2.46 Блок IMAGE\_HEADER

Именно этим блоком по большому счету и отличаются разные версии формата. Назначение полей следующее:

* imageHeaderSize Размер данного (IMAGE\_HEADER) блока в байтах. Равен 40. По этому значению различают версии форматов.
* imageWidth Ширина изображения в пикселях. Всегда положительное число
* imageHeight Высота изображения в пикселях. Особенность заключается в том, что изображение разбивается на горизонтальные строки пикселей. И в случае, если imageHeight положительное, то в блоке PIXEL\_DATA первой будет нижняя, а последней верхняя строка изображения (изображение перевернуто «вверх ногами»). Если задать отрицательное значение высоты (в дополнительном коде), то первой будет верхняя, а последней – нижняя строка изображения.
* mustBeOne Действительно всегда равен одному… В официальной документации это поле называется biPlanes. Т.е. количество граней, размерностей.
* bitPerPixel Количество бит на пиксель.
  + 1,4,8. Когда количество бит на пиксель указано равным 1, 4 или 8, размер палитры равен: 21, 24, 28. Т.е. 2bitPerPixel. При этом палитра (блок PALETTE) присутствует и PIXEL\_DATA содержит bitPerPixel-битные индексы цветов в палитре.
  + 16. Если поле compressionType (см. далее) содержит значение BI\_RGB, то палитра (блок PALETTE) отсутствует. При этом каждое 16 битное число в PIXEL\_DATA хранит интенсивность красной, зелёной и синей компоненты пиксела. Старший бит не используется, и на каждую компоненту отведено 5 бит: 0RRRRRGGGGGBBBBB. Если поле compressionType содержит значение BI\_BITFIELDS, палитра хранит три четырёхбайтовых значения, определяющих маску для каждой из трёх компонент цвета . Каждый пиксель изображения представлен 16 битным числом, из которого с помощью масок извлекаются цветовые компоненты. Последовательности бит каждой компоненты должны следовать непрерывно, не перекрываясь и не пересекаясь с последовательностями других компонент. Следующие маски являются стандартными, т.е. поддерживаются старыми версиями Windows: 5-5-5, где маска синей компоненты 0х001F, зелёной 0x03E0, красной 0x7C00; и 5-6-5, где маска синей компоненты 0x001F, зелёной 0x07E0, красной 0xF800.. В наше время можно использовать любые неперекрывающиеся маски
  + 24. Палитра не используется, каждая тройка байт изображения представляет один пиксель, по байту для интенсивности синего B, зелёного G и красного R канала соответственно.
  + 32. Если поле compressionType (см. далее) содержит значение BI\_RGB, то палитра (блок PALETTE) отсутствует. При этом каждое 32 битное число в PIXEL\_DATA хранит интенсивность красной, зелёной и синей компоненты пикселя. Старший байт не используется, оставшиеся три байта хранят соответственно данные синей B, зеленой G и красной R компонент. Если поле compressionType содержит значение BI\_BITFIELDS, палитра хранит три четырёхбайтовых значения, определяющих маску для каждой из трёх компонент цвета. Каждый пиксель изображения представлен 32 битным числом, из которого с помощью масок извлекаются цветовые компоненты. Последовательности бит каждой компоненты должны следовать непрерывно, не перекрываясь и не пересекаясь с последовательностями других компонент.
* compressionType Тип сжатия.
  + 0 (BI\_RGB). Сжатие не используется.
  + 1 (BI\_RLE8). Используется для 8-битных изображений – сжатие по алгоритму RLE8.
  + 2 (BI\_RLE4). Используется для 4-битных изображений – сжатие по алгоритму RLE4.
  + 3 (BI\_BITFIELDS). Изображение не сжато. Используются маски для выделения компонент из элементов PIXEL\_DATA для изображений с 16 и 32 битами на пиксель (см. выше назначение поля bitPerPixel).
* imageSize Размер изображения в байтах.
* pixelPerMeterX Указывает предпочтительное разрешение в пикселях на метр по горизонтали. Некоторые приложения могут учитывать параметры pixelPerMeterX и pixelPerMeterY. Например, сообразуясь с возможностями устройства вывода изображений.
* pixelPerMeterY Указыват предпочтительное разрешение в пикселях на метр по вертикали.
* usedPaletteItemsCount Содержит количество действительно используемых элементов палитры. Содержит ноль, если используются все цвета палитры.
* importantPaletteItemsCount Указывает количество элементов палитры, необходимых для отображения изображения. Если содержит ноль – все элементы необходимы. Порой это важно для устройств с ограниченными возможностями.

Блок PALETTE. Палитра. Может и отсутствовать (см. выше). Если палитра присутствует, и поле compressionType блока IMAGE\_HEADER содержит значение BI\_BITFIELDS, то блок палитры состоит из трех 4-байтовых масок. При этом на двоичное число, соответствующее значению цвета пикселя, в блоке PIXEL\_DATA по отдельности накладываются маски и извлекаются соответствующие им компоненты цвета. Пример для 16-битных данных приведен на рисунке Рис. 2.47.

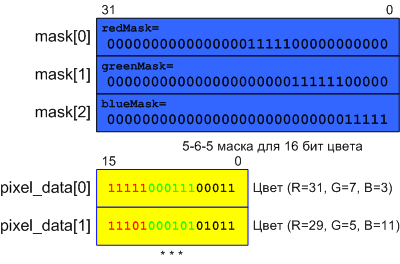


Рис. 2.47 Маски, накладываемые на данные пикселей

Если палитра присутствует, и поле compressionType блока IMAGE\_HEADER содержит значение отличное от BI\_BITFIELDS, то блок палитры состоит из соответствующего количества 4-байтных элементов палитры. Значения вклада (яркости) компонент синего B, зеленого G и красного R цветов сохраняются в первых трех байтах структуры, последний, четвертый байт не используется и содержит ноль. Структура палитры представлена на рисунке на примере 16-битных данных.



Рис. 2.48 Элементы палитры

Блок PIXEL\_DATA. Будем считать, что изображение представлено матрицей пикселей , размером , где – количество строк, а – количество столбцов. В BMP файле этим параметрам соответсвуют поля imageHeight и imageWidth блока IMAGE\_HEADER. Верхний левый пиксел изображения имеет координаты , правый нижний – . Блок PIXEL\_DATA содержит представление матрицы . Представление отдельного пиксела зависит от числа бит на пиксел (поле bitPerPixel блока IMAGE\_HEADER).

* 1 и 4 бита на пиксель. Каждый байт данных содержит либо восемь, либо два битовых поля, содержащих двоичные числа, являющиеся индексами цвета в цветовой палитре (см. Рис. 2.48). Старшее поле в байте относится к крайнему левому пикселю.
* 8 бит на пиксель. Пикселю соответствует байт. Значение байта – индекс в палитре.
* 16 бит на пиксель. Пиксель описывается 2-байтовым целым числом. Если поле compressionType блока IMAGE\_HEADER содержит значение BI\_RGB, то старший бит не используется, и на каждую компоненту отведено 5 бит: 0RRRRRGGGGGBBBBB. Если же поле compressionType содержит значение BI\_BITFIELDS, то три 4-байтные битовые маски, находящиеся в блоке PALETTE, выделяют компоненты красного, зеленого и синего цветов из соответствующего 16-разрядного двоичного числа см. Рис. 2.47 .
* 24 бита на пиксел. Каждый пиксел представлен тремя подряд идущими байтами, которые содержат значения синего, зеленого и красного компонент.
* 32 бита на пиксел. Каждый пиксел описывается 4-х байтным числом. Если поле compressionType блока IMAGE\_HEADER содержит значение BI\_RGB, то пиксел кодируется в трех младших байтах так же как и в случае 24 бит на пиксел, а старший байт не используется. Если же поле compressionType содержит значение BI\_BITFIELDS, то три 4-байтные битовые маски, находящиеся в блоке PALETTE, выделяют компоненты красного, зеленого и синего цветов из соответствующего 32-разрядного двоичного числа см. Рис. 2.47.

В блоке PIXEL\_DATA матрица изображения представлена одномерным массивом байт. Строке матрицы всегда соответствует блок данных в файле, который должен быть кратен четырем байтам. Количество байт на строку в файле определяется по формуле:

Данные нулевого пикселя -й строки (т.е. ) начинаются с -го байта блока PIXEL\_DATA. Как уже было сказано выше, если поле imageHeight блока IMAGE\_HEADER положительно, то -й строке файла будет соответствовать imageHeight- строка изображения , иначе, если imageHeight отрицательно, то -й строке файла соответствует -я строка изображения. Пример расположения изображения 3×3 в блоке PIXEL\_DATA при кодировании 8 бит на пиксель, приведен на рисунке.



Рис. 2.49 Пример хранения изображения в блоке PIXEL\_DATA

Пример бинарного изображения в формате BMP приведен на рисунке Рис. 2.50. В соответствии с этим в шестнадцатеричном редакторе был сформирован двоичный образ файла (см. Рис. 2.51). Образ был сохранен в файл oneBitPerPixel.bmp. Если вы потратите некоторое время на то, чтобы разобраться в приведенном примере, то восстановите на бумаге изображение, созданное когда-то Дуэйном Бибби в том виде, в котором оно отображается программами-просмотрщиками BMP формата (внизу на Рис. 2.45).

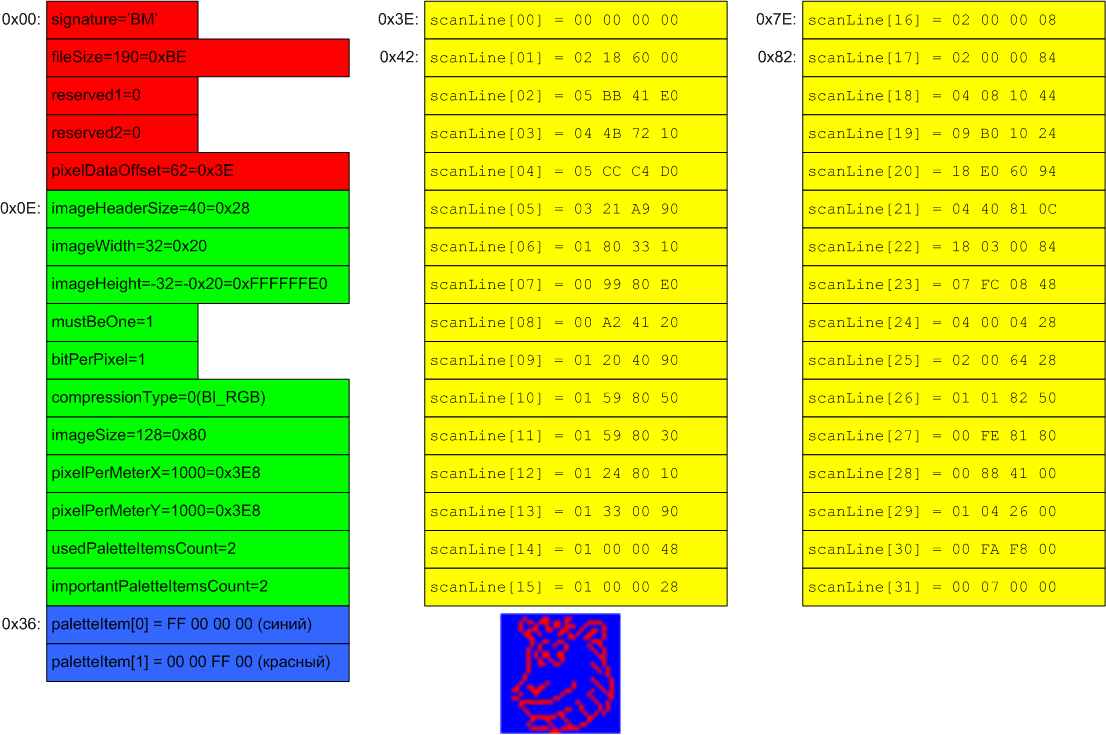


Рис. 2.50 Пример заполненного данными изображения формата BMP

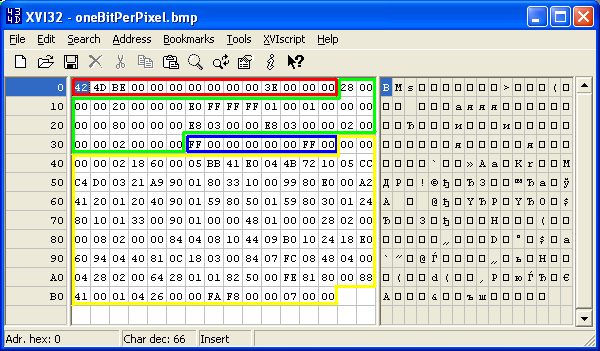


Рис. 2.51 BMP-файл примера на Рис. 2.50 в hex-редакторе

Сжатие. Для представления графических данных требуется большой объем информации. В рассматриваемой версии BMP могут использоваться два простейших алгоритма сжатия: Rle4, Rle8. Мы рассмотрим алгоритм Rle8 и отличия Rle4 от Rle8. RLE расшифровывается как Run-length encoding и переводится как кодирование длин серий. Надо отметить, что данные алгоритмы позволяют сжимать изображения, в которых встречаются достаточно длинные последовательности пикселей одинакового цвета и потому не всегда сжатие с помощью данных алгоритмов приводит к экономии памяти.

Схема кодирования Rle8 приведена на рисунке Рис. 2.52.

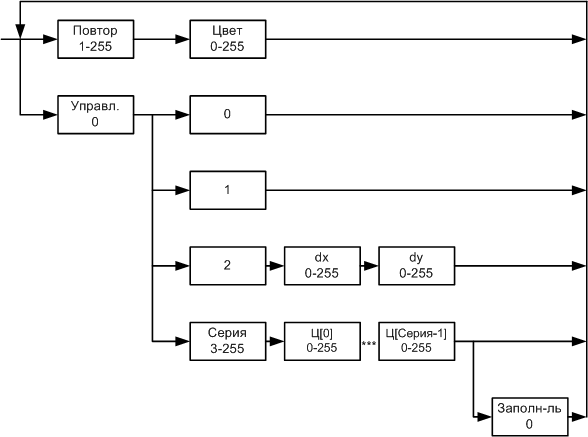


Рис. 2.52 Схема кодирвания Rle8

Итак, по схеме кодирования: в самом начале алгоритма или при очередной итерации первый байт сжатых данных может содержать.

1. Ненулевое значение (Повтор 1-255). Это значение следует трактовать как длину цепочки (серии) байт, состоящей из байт со значением, указанным во втором байте. Значение второго байта это, как уже известно, – индекс цвета в палитре. Т.е. данным двум байтам соответствует цепочка несжатых данных.
2. Ноль (Управл. 0). Это признак управляющего кода и для дальнейшего декодирования важно значение второго байта, которое может быть равным.
   1. Нулю (0). Это означает конец строки пикселей. Остаток строки в изображении будет заполнен цветом фона (0-й цвет).
   2. Единице (1). Это означает конец данных изображения. Остаток будет заполнен цветом фона.
   3. Двойке (2). Это означает, что относительно текущего пикселя в декодированном изображении следует сместиться на dx пикселей по горизонтали и на dy по вертикали. Значения dx, dy находятся в следующих двух байтах. При этом dx, dy беззнаковые, т.е. положительные числа. Направление смещения по вертикали зависит от того, положительное или отрицательное значение находится в поле imageHeight блока IMAGE\_HEADER. Получившийся промежуток в изображении заполняется цветом фона.
   4. Числу от 3 до 255 (Серия 3-255). Это значение следует трактовать как длину цепочки (серии) байт, идущих следом. Байты цепочки содержат индексы цвета в палитре. Данный вариант применяется для хранения блока несжатых данных. Если значение длины серии нечетно, то нужно учитывать то, что серия дополнена нулевыми байтами до четного их количества.

Пример сжатия изображения буквы Ё вышеприведенным алгоритмом приведен на рисунке. Видно, что достигнут определенный выигрыш: 144 байта против 256.

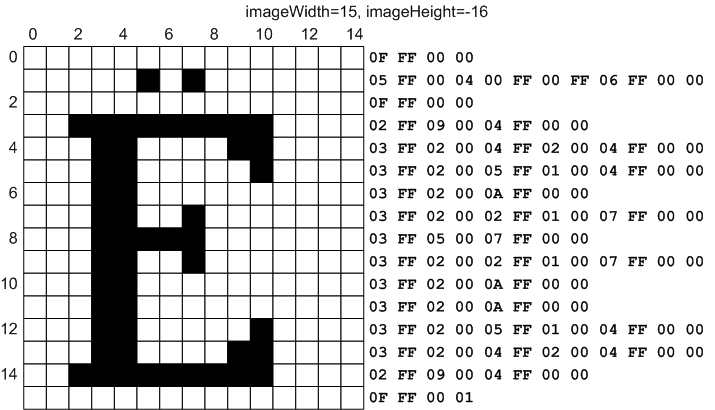


Рис. 2.53 Сжатое алгоритмом Rle8 изображение буквы Ё

Отличия алгоритма Rle4 от Rle8 незначительны. Например, если первый байт декодируемой структуры не ноль, то далее следует байт, который содержит *два* четырехбитных индекса цвета. То есть в данном случае могут кодироваться как серии пикселей одинакового цвета, так и серии одинаковых пар пикселей разного цвета:

|  |
| --- |
| Закодировано Раскодировано  01 AB A  02 AB A B  05 AA A A A A A  05 AB A B A B A  06 AB A B A B A B |

В отношении управляющих последовательностей, начинающихся с нулевого байта, Rle4 также имеет единственное отличие от Rle8: если вторым после нулевого управляющего идет байт, содержащий значения в диапазоне [3,255], то серия несжатых данных содержит именно такое количество *тетрад*. Серия сжатых данных должна содержать четное количество *байт* поэтому до указанного количества серия дополняется нулевыми *нибблами*:

|  |
| --- |
| Закодировано Раскодировано  00 03 AB C**0** A B C  00 04 AB CD A B C D  00 05 AB CD E**0 00** A B C D E  00 06 AB CD EF **00** A B C D E F |

В заключение хотелось бы отметить, что более новые версии формата BMP поддерживают более совершенные алгоритмы сжатия изображений такие как JPEG (с потерями) и PNG (без потерь). Впрочем, из-за этого сам по себе *формат* изменяется совсем незначительно.

#### Формат XBM

Этот растровый формат позволяет сохранять лишь монохромные изображения. Данные изображения сохраняются в текстовом формате. Он активно использовался для непосредственного задания иконок в исходных текстах программ на языке C при создании программ под графическую систему **X**-window операционных систем семейства Unix. XBM сокращение от **X** **B**it**M**ap. Итак, XBM это корректный текст на языке программирования C:

|  |
| --- |
| #define myimage\_height 32  #define myimage\_width 32  static unsigned char myimage\_bits[]={  0x00,0x00,0x00,0x00, 0x40,0x18,0x06,0x00, 0xa0,0xdd,0x82,0x07,  0x20,0xd2,0x4e,0x08, 0xa0,0x33,0x23,0x0b, 0xc0,0x84,0x95,0x09,  0x80,0x01,0xcc,0x08, 0x00,0x99,0x01,0x07, 0x00,0x45,0x82,0x04,  0x80,0x04,0x02,0x09, 0x80,0x9a,0x01,0x0a, 0x80,0x9a,0x01,0x0c,  0x80,0x24,0x01,0x08, 0x80,0xcc,0x00,0x09, 0x80,0x00,0x00,0x12,  0x80,0x00,0x00,0x14, 0x40,0x00,0x00,0x10, 0x40,0x00,0x00,0x21,  0x20,0x10,0x08,0x22, 0x90,0x0d,0x08,0x24, 0x18,0x07,0x06,0x29,  0x20,0x02,0x81,0x30, 0x18,0xc0,0x00,0x21, 0xe0,0x3f,0x10,0x12,  0x20,0x00,0x20,0x14, 0x40,0x00,0x26,0x14, 0x80,0x80,0x41,0x0a,  0x00,0x7f,0x81,0x01, 0x00,0x11,0x82,0x00, 0x80,0x20,0x64,0x00,  0x00,0x5f,0x1f,0x00, 0x00,0xe0,0x00,0x00  }; |

В первых строках файла директивами препроцессора задается количество точек по высоте (myimage\_height) и по ширине (myimage\_width), далее определяется массив байт (myimage\_bits), содержащий битовую карту изображения (единичному биту соответствует чёрный цвет, а нулевому – белый). Изображение формируется по горизонтальным строкам. Количество строк – высота изображения myimage\_height. Количество бит для представления строки line\_size всегда кратно восьми (байту) и определяется по формуле:

Лишние биты просто не используются. Общий размер массива будет составлять: .

Особенностью формата является то, что самой левой из восьми точек изображения, кодируемых байтом, соответствует младший разряд соответствующего байту двоичного числа (т.е. разряды двоичного числа придется записать в обратном привычному порядке). Потрудитесь над примером с карандашом и листком бумаги, и вы восстановите пиктограмму для классической иллюстрации Дуэйна Бибби из руководства пользователя по TeX (см. Рис. 2.54).



Рис. 2.54 Одна из иллюстраций к руководству по TeX (LaTeX)

Файлы в формате XBM имеют расширение \*.xbm и открываются многими программами-просмотрщиками изображений. В частности на момент написания статьи xbm-файл открывался web-браузерами Google Chrome 5.0 и Opera 10.51, современные версии Microsoft Internet Explorer и Mozilla Firefox от поддержки устаревшего формата отказались.

#### Формат TIFF

TIFF (англ. Tagged Image File Format) - бинарный формат хранения растровых графических изображений. Изначально был разработан компанией Aldus в сотрудничестве с Microsoft. В настоящее время правообладателем спецификаций формата является Adobe. Является стандартом для хранения высококачественных изображений, активно поддерживается сканерами и факсами. Файлы данного формата, как правило, имеют расширение \*.tiff или \*.tif. Формат позволяет сохранять бинарные, монохромные, полноцветные и индексированные изображения в моделях RGB, CMYK, YCbCr, CIE Lab... Мы рассмотрим формат шестой ревизии: TIFF Revision 6.0.

Итак, в самом начале файла идет 8-и байтный заголовок (Header), содержащий смещение от начала файла первой структуры, которая называется *директорией файла изображения* (IFD - image file directory). IFD содержит необходимое количество структур IFD Entry, а также смещение следующей IFD. Каждая IFD описывает отдельное изображение. Соответственно один TIFF файл может содержать несколько изображений, хотя некоторые графические приложения загружают данные только первой IFD. Элемент IFD – IFD Entry содержит в своем составе тег (откуда и название формата), определяющий функциональное назначение элемента. Тег содержит число, и в пределах IFD элементы отсортированы по возрастанию значений тегов. Структура TIFF файла приведена на Рис. 2.55.

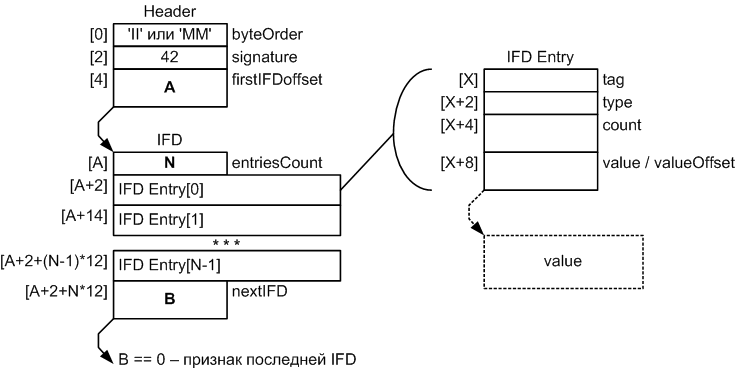


Рис. 2.55 Структура TIFF файла

Рассмотрим форматы каждой структуры подробнее. Начнем с Header. Это восьмибайтная структура следующего вида.

* Байты 0-1. byteOrder. Содержит ASCII строку «II» или «MM» (соответственно 0x4949 или 0x4D4D). Определяет little-endian («II») или big-endian («MM») способ представления многобайтных чисел в файле. На соответствующей архитектуре это позволяет сэкономить массу времени на перестановки байт. Т.е. все последующие значения в структурах данных будут представлены так, как указано в данном поле.
* Байты 2-3. signature. Поле содержит число-сигнатуру, по которой можно идентифицировать TIFF файл (наряду с полем byteOrder). Значение сигнатуры: 42.
* Байты 4-7. firstIFDoffset. Содержит смещение относительно начала файла первой структуры IFD.

Теперь об IFD. IFD описывает одно из изображений, хранимых в файле. В TIFF файле должна быть по крайней мере одна структура IFD. IFD - переменного размера структура следующего вида.

* Байты 0-1. entriesCount. Содержит количество следующих за данным полем 12-байтных IFD Entry. Это количество не может быть нулевым. Допустим, что у нас в этом поле записано целое положительное число N. N>0.
* Байты со 2-го по (1+N\*12)-й. Содержат 12 байтные структуры IFD Entry в количестве N штук.
* Байты с (2+N\*12)-го по (3+N\*12)-й. nextIFD. Содержит смещение относительно начала файла до следующей структуры IFD. Если это поле содержит ноль, то данная структура IFD является последней в списке.

И наконец, IFD Entry. Элемент структуры IFD. Описывает некоторую *характеристику* изображения. Содержит *признак характеристики* (tag) её значение (value) и тип (type). В пределах IFD должны быть отсортированы по возрастанию значений поля tag. IFD Entry – 12-байтная структура следующего вида.

* Байты 0-1. tag. По этому числу определяется назначение элемента IFD Entry. То есть определяет то, значение какой именно характеристики изображения содержит данный IFD Entry. Далее будут рассмотрены основные значения этого поля.
* Байты 2-3. type. Тип хранимого значения характеристики изображения (поле value). Выделены следующие типы, каждому из которых соответствует число.
  + 1 - BYTE. 8-битное беззнаковое целое число.
  + 2 - ASCII. ASCII-строка 8-битных символов. Последний символ строки должен быть нулевым. Вообще IFD Entry может в качестве значения хранить несколько строк, каждая из которых заканчивается нулем. При этом поле count должно хранить совокупный размер строк в байтах с учетом завершающих нулей.
  + 3 - SHORT. 16-битное беззнаковое целое.
  + 4 - LONG. 32 битное беззнаковое целое.
  + 5 - RATIONAL. Рациональное число. Дробь. Состоит из двух идущих друг за другом значений LONG. Первое – числитель, второе – знаменатель.
  + 6 - SBYTE. 8-битное целое число со знаком.
  + 7 - UNDEFINED. Признак бинарных данных формат которых не определен или зависит от поля tag. Фактически 8-бит.
  + 8 - SSHORT. 16-битное целое число со знаком.
  + 9 - SLONG. 32-битное целое число со знаком.
  + 10 - SRATIONAL. Рациональное число. Дробь. Состоит из двух идущих друг за другом значений SLONG. Первое --- числитель, второе --- знаменатель.
  + 11 - FLOAT. Вещественное число одинарной точности (4-байта) IEEE формат.
  + 12 - DOUBLE. Вещественное число одинарной точности (8-байт) IEEE формат.
  + Байты 4-7. count. Количество значений указанного типа type. Фактически значениями IFD Entry являются массивы.
* Байты 8-11. value / valueOffset. Поле содержит либо непосредственно значение, если оно умещается в 4-байта (значение (или несколько значений) хранится в младших байтах поля), либо смещение относительно начала файла, по которому хранятся значения. Так, например, если указано type=BYTE, count=3, то в младших байтах поля value будут хранится три байта значения. Если, например, type=BYTE, count=5, то поле value / valueOffset содержит смещение в файле, по которому будут храниться значения этих 5 байт данных.

Приведем описание некоторых наиболее важных IFD Entry.

* tag=254(0x00FE)=NewSubfileType. type=LONG. count=1. Тип данных, хранящихся в этом файле. Этот тег является заменой тегу SubfileType, и является очень полезным, когда в одном TIFF файле хранится несколько изображений. Возможные значения value определяются комбинацией независимых бит. Нулевой бит установлен в 1 – изображение является уменьшенной версией другого изображения в файле (1 – в противном случае). Бит 1 установлен в 1 – это одна из страниц многостраничного изображения (1 – в противном случае). Бит 2 установлен в 1 – изображение представляет собой маску прозрачности для другого изображения в этом же файле (используются дополнительные параметры для идентификации какого же именно).
* tag=256=0x0100=ImageWidth. type=LONG. count=1. value – количество колонок в изображении.
* tag=257=0x0101=ImageLength. type=LONG. count=1. мфдvalue – количество строк в изображении.
* tag=259=0x0103=Compression. type=SHORT. count=1. value – используемый способ сжатия данных. Поддерживаются, например следующие виды компрессии: без компрессии (0), CCITT Group 3 (2), CCITT T.4 (3), CCITT T.6 (4), LZW (5), JPEG (6), PackBits (32773).
* tag=262=0x0106=PhotometricInterpretation. type=SHORT. count=1. value – используемая цветовая модель. Черно-белое (0,1), RGB (2), палитра (3), маска прозрачности (4), CMYK (5), YCbCr (6), CIE Lab (8).
* tag=273=0x0111=StripOffsets. type=SHORT или LONG. Смещение для каждой полосы изображения в байтах. Полоса обычно содержит сразу несколько строк.
* tag=278=0x0116=RowsPerStrip. type=SHORT или LONG. Количество строк на полосу.
* tag=279=0x0117=StripByteCounts. type=SHORT или LONG. Количество байт на полосу после компрессии.
* tag=282=0x011A=XResolution. type=RATIONAL. Количество пикселей на единицу ResolutionUnit строки.
* tag=283=0x011B=YResolution. type=RATIONAL. Количество пикселей на единицу ResolutionUnit столбца.
* tag=296=0x0128=ResolutionUnit. type=SHORT. Разрешение данных, хранящихся в XResolution, YResolution. Дюймы (1), сантиметры (3), безразлично (0). Если отсутствует в IFD – по умолчанию дюймы.
* tag=305=0x0141=Software. type=ASCII. Имя и версия создавшего программного продукта.
* tag=306=0x0142=DateTime. type=ASCII. count=20. Дата и время создания изображения в формате <<YYYY:MM:DD HH:MM:SS>>.

Пример TIFF файла, содержащего черно-белое изображение (фактические данные полос не указаны):

|  |
| --- |
| Смещение Описание Значение(value/valueOffset)  Header:  0000 Byte Order 4D4D  0002 42 002A  0004 firstIFDoffset 00000014  IFD:  0014 entriesCount 000C  0016 NewSubfileType 00FE 0004 00000001 00000000  0022 ImageWidth 0100 0004 00000001 000007D0  002E ImageLength 0101 0004 00000001 00000BB8  003A Compression 0103 0003 00000001 80050000  0046 Photometric  Interpretation 0106 0003 00000001 00010000  0052 StripOffsets 0111 0004 000000BC 000000B6  005E RowsPerStrip 0116 0004 00000001 00000010  006A StripByteCounts 0117 0003 000000BC 000003A6  0076 XResolution 011A 0005 00000001 00000696  0082 YResolution 011B 0005 00000001 0000069E  008E Software 0131 0002 0000000E 000006A6  009A DateTime 0132 0002 00000014 000006B6  00A6 nextIFDoffset 00000000  Values longer than 4 bytes:  00B6 StripOffsets Offset0,Offset1,...Offset187  03A6 StripByteCounts Count0, Count1,... Count187  0696 XResolution 0000012C 00000001  069E YResolution 0000012C 00000001  06A6 Software "PageMaker 4.0"  06B6 DateTime "1988:02:18 13:59:59"  Image Data:  00000700 Compressed data for strip 10  xxxxxxxx Compressed data for strip 179  xxxxxxxx Compressed data for strip 53  xxxxxxxx Compressed data for strip 160  .  . |

В заключение можно отметить, что существует развитие формата TIFF, для хранения больших изображений – BigTIFF. Размер TIFF-файла ограничен четырьмя гигабайтами (т.е. ), а BigTIFF ограничивает размеры файла 16 эксбибайтами (т.е. ). На сайте популярной библиотеки для работы с BigTIFF форматом (на момент написания: http://www.aperio.com/bigtiff/) вы найдете примеры изображений размером 1095630 на 939495 пикселей и размером файла в 143 гигабайта. Такие изображения находят применение в медицине, аэро или космической съёмке и во многих других областях. В целом формат изменился не сильно: смещения в файле теперь 64 битные.

#### Формат SVG

SVG (от англ. Scalable Vector Graphics --- масштабируемая векторная графика) – язык разметки масштабируемой векторной графики, созданный Консорциумом W3C и входящий в подмножество расширяемого языка разметки XML, предназначен для описания двумерной векторной и смешанной векторно/растровой графики в формате XML. Поддерживает как неподвижную, так анимированную и интерактивную графику. SVG – открытый стандарт. Разрабатывается с 1999 года, в 2001 году вышла 1.1 версия, которая остается актуальной до сегодняшнего дня, на момент написания данного текста в активной разработке находится версия 1.2.

Документы SVG непосредственно открываются любым современным web-браузером, а также многими программами для работы с векторной графикой.

Документ SVG – это специализированное описание графических данных на языке XML. XML уже обсуждался в параграфе 2.7.2. Имена элементов и их атрибутов насыщены специфичным смысловым содержанием и определяют составляющие изображение графические примитивы, цвета, градиенты, заливки и элементы анимации. Программа, отображающая SVG изображения, интерпретирует соответствующие значения элементов и атрибутов и формирует на их основе изображение.

Итак, следующий XML определяет изображение на Рис. 2.56 в SVG формате. Достаточно сохранить этот текст в текстовом файле с расширением \*.svg и открыть его web-браузером. Внося изменения в исходный файл и обновляя браузер можно добиться понимания назначения элементов и их атрибутов.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <svg version="1.1"  baseProfile="full"  xmlns = "http://www.w3.org/2000/svg"  height = "400"  width = "400">    <rect x="0" y="0"  width="400" height="400"  fill="none" stroke="black"  stroke-width="5" stroke-opacity="0.5"/>    <g fill-opacity="0.6" stroke="black" stroke-width="0.5">    <circle cx="200" cy="200" r="100"  fill="red"  stroke="green" stroke-width="10"  transform="translate(0,-50)" />    <rect x="100" y="100"  height="200" width="200"  fill="blue"  stroke="red" stroke-width="10"  transform="translate(70,60)" />    <polygon fill="lime"  stroke="blue" stroke-width="10"  points="150,100 250,100 300,150 300,250  250,300 150,300 100,250 100,150"  transform="translate(-70,40)"/>    <text x="10" y="30"  fill="yellow"  stroke="red" stroke-width="1"  font-size="22pt">  Пример SVG изображения!!!  </text>  </g>  </svg> |

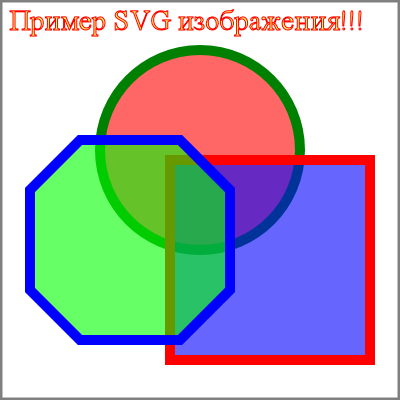


Рис. 2.56 Пример SVG изображения

Не составляет труда разобраться в приведенном XML и соотнести его с изображением. Обратитесь к официальной документации по адресу: <http://www.w3.org/Graphics/SVG/> за дополнительными знаниями (SVG становится частью новой версии HTML – HTML5). В заключение следует отметить, что SVG позволяет создавать анимированные и интерактивные изображения-приложения. Из недостатков можно отметить достаточно большой объем файлов. Проблема решается обычным архивированием с помощью архиватора gzip. XML данные хорошо сжимаются. Сжатые файлы SVG документов получают расширение \*.svgz.

**Вопросы**

1. Какие бывают изображения?
2. Какие виды графики вы знаете? Каковы их особенности?
3. Каким образом происходит восприятие цвета?
4. Раскройте суть эксперимента Германа Гюнтера Грассмана
5. Что такое цветовая модель?
6. Расскажите о модели CIE-RGB? Область её применения?
7. Расскажите о модели CMYK? Какова область её применения?
8. Расскажите о модели CMYK? Какова область её применения?
9. Расскажите о модели CIE-RGB? Какова область её применения?
10. Зачем нужна абстрактная модель CIE-XYZ? Что такое гамут?

**Практические задания**

1. Выполните переход от одной модели к другой (соответствующие компоненты цвета представлены числами от 0 до 1):

|  |  |  |
| --- | --- | --- |
| Вариант | Исходные параметры модели | Целевая модель |
| 1 | CIE-RGB=(0.6, 0.1, 0.9) | CMYK |
| 2 | CIE-RGB=(0.7, 0.9, 0.3) | YCbCr |
| 3 | CIE-RGB=(0.7, 0.3, 0.9) | CIE-XYZ |
| 4 | CMYK=(0.4, 0.3, 0.9, 0.1) | CIE-RGB |
| 5 | CMYK=(0.5, 0.1, 0.3, 0.9) | CIE-RGB |
| 6 | CIE-XYZ=(0.7, 0.1, 0.8) | CIE-RGB |

1. Для модели CIE-XYZ Известно два хроматических числа , и коэффициент . Восстановите коэффициенты , и хроматическое число .

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант |  |  |  |
| 1 | 0.5 | 0.1 | 7 |
| 2 | 0.4 | 0.3 | 5 |
| 3 | 0.7 | 0.2 | 9 |
| 4 | 0.3 | 0.1 | 3 |
| 5 | 0.1 | 0.5 | 2 |
| 6 | 0.2 | 0.4 | 8 |

1. Сформируйте и сожмите изображение пикселя с помощью одного из алгоритмов RLE.

|  |  |  |
| --- | --- | --- |
| Вариант | Изображение | Алгоритм |
| 1 | ☠ | Rle4 |
| 2 | ♣ | Rle8 |
| 3 | ♥ | Rle4 |
| 4 | ☯ | Rle8 |
| 5 | ☛ | Rle4 |
| 6 | ☥ | Rle8 |

1. Сформируйте изображения следующих символов в указанном формате.

|  |  |  |
| --- | --- | --- |
| Вариант | Изображение | Формат |
| 1 | ☎ | BMP, бинарное |
| 2 | ☭ | BMP, rle8 |
| 3 | ☂ | BMP, rle4 |
| 4 | ✏ | XBM |
| 5 | ✄ | TIFF,без сжатия |
| 6 | ☆ | SVG |

1. Сформируйте цветные изображения следующих символов в формате SVG (можно моделировать точки растра примитивами, или полнее изучить теги SVG).

|  |  |
| --- | --- |
| Вариант | Изображение |
| 1 | ❄ |
| 2 | ☄ |
| 3 | ♂ |
| 4 | ☀ |
| 5 | ☮ |
| 6 | ✬ |

### Форматы аудио данных

С интуитивной точки зрения *звук* – это ощущения, *воспринимаемые* нашим ухом и *интерпретируемые* мозгом определенным образом. С научной – звук – это возмущение среды, распространяющееся с помощью волн давления. Поэтому мы будем рассматривать определенные характеристики звука с научной точки зрения, а также поговорим об особенностях восприятия и интерпретации звука человеком.

Звук можно рассматривать как продольную волну. То есть волну, в которой направления *возмущения* совпадают с направлением её *распространения*. Основные характеристики волны это её *амплитуда* (A), *скорость распространения* и *период* возмущений (T). Частота звука – это количество *периодов возмущения* в единицу времени (1/T). Частота измеряется в герцах (Гц или Hz).

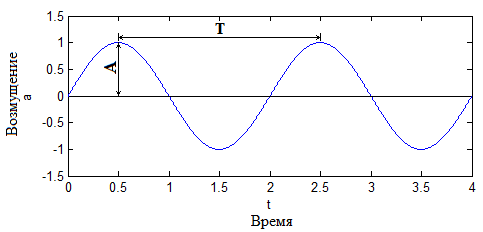


Рис. 2.57 Звук как волна

Человеческое ухо способно воспринимать звуковые волны с частотой от 20 до 22000 герц. Это зависит от здоровья и возраста человека. Указанный диапазон называется диапазоном слышимых частот. Слышимых именно человеком. Например, собаки прекрасно слышат звук с частотой больше 22000 герц, такой звук называется *ультразвуком*.

Амплитуда колебаний воспринимается как *громкость*. Человеческое ухо воспринимает достаточно широкий диапазон амплитуд. Поэтому громкость (мощность) звука обычно измеряют в логарифмической шкале, в *децибелах* (дБ или dB). Например, громкость комариного писка и громкость выстрела пушки различаются на 11-12 порядков. Т.е. если писк комара это 1, то выстрел пушки это . В логарифмической шкале (основание 10) 1 будет соответствовать 0 (), а всего лишь 11 (). То есть диапазон изменения величин будет вместо . Это гораздо удобнее. За минимальный уровень мощности звука взят порог слышимости в микроватт, то есть ватта. Соответственно уровень в децибелах будет определяться так:

В этой формуле – это уровень мощности, измеренный в ваттах. Например, устройство, производящее 1 ватт акустической мощности имеет уровень громкости

Примерная шкала оценки уровней звука в децибелах приведена на Рис. 2.58. Кстати, значения в децибелах нельзя складывать. Например, если прямо над головой счастливца, любующегося ниагарским водопадом (100 дБ), грянул гром (тоже около 100 дБ), то турист насладился звуком «всего» в децибела. Удвоение мощности звука увеличивает уровень звука примерно на три децибела.



Рис. 2.58 Оценки уровней звука в децибелах

Звук разной частоты воспринимается человеком по-разному. Наиболее чутко ухо улавливает звуки с частотой от 2000 до 4000 герц. Приблизительный график порога чувствительности человека приведен на Рис. 2.59. Звуки, уровень громкости которых находится ниже приведенной кривой, не воспринимаются.

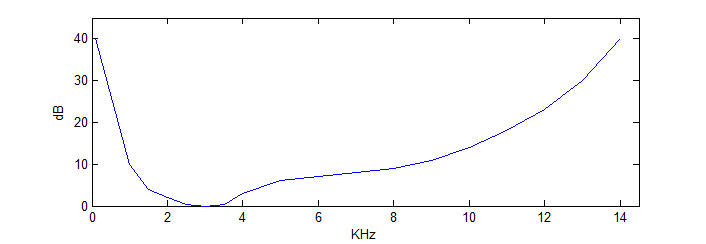


Рис. 2.59 Зависимость порога слышимости (децибелы dB) от частоты (килогерцы KHz)

Не только частота звука влияет на восприятие звука. Существует особенность слуха, называемая *частотное маскирование*. Нормально слышимый звук может оказаться неслышимым, если накрывается *в то же время* более громким звуком, близким по частоте. Порог слышимости как бы приподнимается этим накрывающим громким звуком в некоторой окрестности частот. Пример приведен на Рис. 2.60.

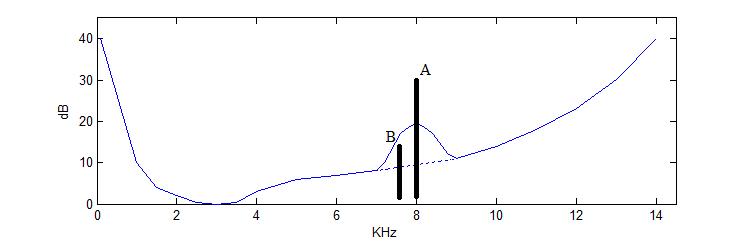


Рис. 2.60 Частотное маскирование

Громкий звук A с частотой 8 KHz «приподнял» порог слышимости (см. Рис. 2.59), и менее громкий звук B с частотой 7.6 KHz стал не слышен.

Другая особенность, называемая *временным маскированием*, заключается в том, что громкий звук A сделает неслышимым *следующий за ним* более слабый звук B, близкий по частоте. Пример приведен на Рис. 2.61.

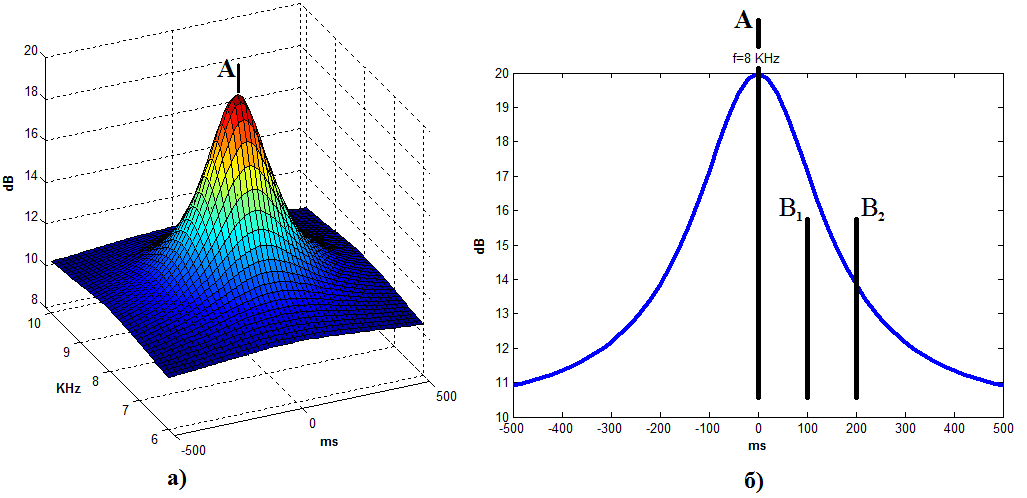


Рис. 2.61 Временное маскирование

Как видно, если оба звука A и B происходят в момент t=0 ms, мы имеем частотное маскирование (именно поэтому приведен трехмерный график участка порога слышимости (добавлена ось времени)). Допустим теперь, что два звука и имеют одинаковую громкость и частоту (8 KHz). следуют после того, как в момент времени t=0 ms прозвучал громкий звук A той же частоты. Звук , следующий через 100 ms не будет услышан, а звук , следующий 200 ms будет.

*Частотное* и *временное маскирование* активно используются многими алгоритмами сжатия звука с потерями.

Мы уже упоминали о *звуке*, когда обсуждали различия между *аналоговым* и *цифровым* *сигналами*. Далее мы будем обсуждать *оцифрованный* звук. Чтобы оцифровать звук, требуется в определенные моменты времени выполнить замеры значений уровня его громкости. Отдельное значение называют сэмплом, а сам процесс – сэмплированием. Числовой ряд значений-сэмплов и будет *цифровым* представлением звука:



Рис. 2.62 Сэмплирование

Устройство *сэмплирования* называется цифро-аналоговым преобразователем – ЦАП (или DAC –Digital to Analog Converter). Ответ на вопрос, с какой частотой выполнять замеры значений (сэмплирование) без потерь качества, дает *теорема Котельникова*. В некоторых источниках эту теорему называют *теоремой Найквиста*, иногда, отдавая дань уважения обоим ученым, параллельно сформулировавшим теорему, её называют *теоремой Найквиста-Котельникова*. Согласно *теореме* *Найквиста-Котельникова*, сэмплирование следует делать с частотой, не меньшей удвоенной частоты волн данного звука. Диапазон слышимых частот лежит в интервале 20-22000 герц. Если необходимо получить звук высокого качества, сэмплирование нужно проводить с частотой 44000 герц. Меньшие частоты приводят к искажениям, например, при сэмплировании с частотой 8 КГц можно легко спутать согласные «с» и «ф». Указанный способ сэмплирвания называется *импульсной кодовой модуляцией* (PCM – Pulse Code Modulation). *Кодовой модуляцией* называется техника перевода аналогового сигнала в цифровую форму. Кроме PCM существуют еще несколько техник: импульсная амплитудная модуляция PAM (Pulse Amplitude Modulation), импульсная позиционная модуляция PPM (Pulse Position Modulation), импульсная числовая модуляция PNM (Pulse Number Modulation). На практике PCM показывает большую эффективность.

#### Формат WAV

Формат создан тандемом фирм Microsoft и IBM для работы с *оцифрованным* звуком. В основе лежит более общий бинарный формат RIFF (Resource Interchange File Format – формат файла для обмена ресурсами). RIFF использует little-endian порядок байт. Основным строительным блоком RIFF является chunk (фрагмент, кусок, блок и т.д. данных) имеющий следующую базовую структуру:



Рис. 2.63 Базовая структура блока (chunk) RIFF

Поля имеют следующее назначение.

* chunkId – идентификатор блока. Содержит ASCII строку из четырех символов. Иногда значение трактуется как четырехбайтное целое, например, 'RIFF' (0x46464952), 'LIST' (0x5453494C). По значению данного поля определяют, как интерпретировать полезную нагрузку в поле данных chunkData.
* chunkSize – размер поля данных в байтах.
* chunkData – полезные данные в количестве chunkSize байт. Могут содержать вложенные chunk-блоки.

Chunk в файле идут подряд друг за другом, поэтому, зная размер chunkSize поля данных текущего chunk, всегда можно найти начало chunk, следующего за ним. Далее для формата WAV определяется несколько специфичных chunk (обратите внимание, что все специфичные chunk на самом деле являются базовой структурой блока). Chunk заголовка файла имеет идентификатор chunkId='RIFF' и определяется так:



Рис. 2.64 Структура chunk заголовка RIFF WAV файла

Заголовок помимо стандартных полей в поле данных содержит идентификатор типа файла fileType (ASCII строка из четырех символов), равный для WAV файлов 'WAVE' (0x57415645). Поле fileData содержит вложенные chunk, размечающие полезные данные файла.

Вторым важнейшим (и обязательным) chunk является блок форматирования, имеющий следующую структуру:



Рис. 2.65 Блок описания формата аудио данных WAV файла ('fmt ')

Как видно поле chunkId содержит значение 'fmt ' (0x20746D66). Поля имеют следующее назначение.

* chunkSize – содержит размер поля данных chunkData блока формата аудио данных. В случае, если необязательные поля extraParamsSize и extraParams отсутствуют, то содержит значение 16.
* compressionCode – содержит код, определяющий используемый тип сжатия, и, следовательно, формат аудио данных полезной нагрузки в данном файле (см. далее описание фрагмента данных chunkId='data'). Наиболее распространенные коды следующие:

|  |  |
| --- | --- |
| compressionCode | Используемый формат |
| 0 (0x0000) | Формат неизвестен |
| 1 (0x0001) | PCM/не сжатые данные |
| 2 (0x0002) | Microsoft ADPCM |
| 6 (0x0006) | ITU G.711 a-law |
| 7 (0x0007) | ITU G.711 Âµ-law |
| 17 (0x0011) | IMA ADPCM |
| 20 (0x0016) | ITU G.723 ADPCM (Yamaha) |
| 49 (0x0031) | GSM 6.10 |
| 64 (0x0040) | ITU G.721 ADPCM |
| 80 (0x0050) | MPEG |
| 65535 (0xFFFF) | Experimental |

* numChannels – определяет сколько независимых аудио сигналов (каналов) содержится в данном файле. Значение 1 – один канал (моно звук), 2 – стереозвук, 3 – и т.д.
* sampleRate – количество воспроизводимых сэмплов в секунду. Частота сэмплирования. Не зависит от числа каналов. Популярные значения: 8000 (8kHz), 22000(22kHz), 44100(44kHz),…
* byteRate – данное значение определяет, сколько байт в секунду нужно подавать на устройство воспроизведения. Рассчитывается как sampleRate\*blockAlign.
* blockAlign – количество байт, необходимых для хранения сэмпла с учетом количества каналов: .
* bitsPerSample – количество значащих бит в сэмпле. Обычно 8, 16, 24, 32.
* extraParamsSize, extraParams – эти поля могут отсутствовать. Это зависит от значения поля compressionCode. Например, когда данные не сжаты (compressionCode=1) эти поля отсутствуют. Для других форматов могут потребоваться дополнительные параметры, сохраняемые в поле произвольной длины extraParams. Количество байт в extraParams указывается в поле extraParamsSize.

Собственно аудио данные сохраняются в chunk с идентификатором chunkId='data'. В каком формате они находятся, определяется полем compressionCode блока 'fmt '. Если данные не сжаты, то есть compressionCode=1, то сэмплы сохраняются так: сначала сохраняются первые сэмплы каждого канала, упорядоченные по номеру канала, затем вторые и т.д.

Блоки с идентификаторами fmt и data вложены в блок заголовка:



Рис. 2.66 Общая структура WAV файла (необходимый минимум)

Пример WAV файла, открытого в шестнадцатеричном редакторе приведен далее.

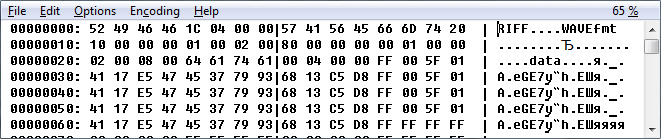


Рис. 2.67 WAV файл открыт в hex-редакторе

Соотнести структуру не составляет труда:

|  |
| --- |
| 'RIFF'('WAVE',  52 49 46 46 |chunkId='RIFF'  1c 04 00 00 |fileSize=0x41c(1052)  57 41 56 45 |fileType='WAVE'    'fmt '(    66 6d 74 20 |chunkId='fmt '  10 00 00 00 |chunkSize=0x10(16)  01 00 |compressionCode=0x1  02 00 |numChannels=0x2  80 00 00 00 |sampleRate=0x80(128)  00 01 00 00 |byteRate=0x100(256)  02 00 |blockAlign=0x2  08 00 |bitsPerSample=0x8  )    'data'(    64 61 74 61 |chunkId='DATA'  00 04 00 00 |chunkSize=0x400(1024)  ff 00 |<channel0:sample0>,<channel1:sample0>  5f 01 |<channel0:sample1>,<channel1:sample1>  41 17 |<channel0:sample2>,<channel1:sample2>  e5 47 |<channel0:sample3>,<channel1:sample3>  45 37 |<channel0:sample4>,<channel1:sample4>  79 93 |<channel0:sample5>,<channel1:sample5>  ... |etc...  )  ) |

Конечно, приведенное описание не является полным. WAV формат определяет множество специфичных chunk.

**Вопросы**

1. Что такое звук? Назовите основные характеристики звука.
2. Как воспринимается амплитуда, частота. Как скорость звука связана с частотой и длиной волны?
3. Что такое частотное маскирование?
4. Что такое временное маскирование?
5. Что такое сэмплирование?
6. Что утверждает теорема Найквиста-Котельникова?
7. Какой объем данных занимает одна минута 32 битного стереозвука с частотой сэмплирования 44 КГц.

**Задания**

1. Скольким децибелам соответствует мощность звука:

|  |  |
| --- | --- |
| Вариант | Мощность, Ватт |
| 1 | 10 |
| 2 | 20 |
| 3 | 300 |
| 4 |  |
| 5 | 100 |
| 6 | 0.001 |

1. Одновременно прозвучали два звука громкостью A и B децибелл соответственно. Какова громкость звука воспринимаемого слушателем?

|  |  |  |
| --- | --- | --- |
| Вариант | Громкость A | Громкость B |
| 1 | 10 дБ | 20 дБ |
| 2 | 20 дБ | 20 дБ |
| 3 | 100 дБ | 10 дБ |
| 4 | 100 дБ | 110 дБ |
| 5 | 10 дБ | 10 дБ |
| 6 | дБ | 1 дБ |

1. Сформируйте данные в формате wav без сжатия (не указывая сэмплы в 'data' chunk) для звука продолжительностью в 2 секунды, если заданы следующие параметры:

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Частота сэмплировния | Количество каналов | Количество бит на сэмпл |
| 1 | 200 Гц | 4 | 8 |
| 2 | 1000 Гц | 2 | 16 |
| 3 | 8000 Гц | 3 | 32 |
| 4 | 8100 Гц | 2 | 16 |
| 5 | 8500 Гц | 3 | 8 |
| 6 | 44000 Гц | 2 | 32 |

### Форматы видео данных

Зрение человека обладает некоторой инертностью восприятия. Чтобы создать иллюзию непрерывного движения на экране, требуется сделать моментальные «снимки» (кадры) происходящего с частотой как минимум 24 кадра в секунду и воспроизвести их в том же порядке и с той же частотой на экране. Скорость смены кадров обычно обозначают fps (frame per second). Иллюзия будет полнее, если сопроводить смену кадров на экране *звуком*. Индустрия кино шагнула далеко вперед и в некоторых кинотеатрах задействуются и другие (кроме зрительных и слуховых) органы чувств, например, осязание, чувство равновесия, слух и вкус. Но это уже изыски.

Итак, упрощенно говоря, видео представляет собой один или несколько *потоков* изображений, и один или несколько звуковых *потоков*, синхронизированных по времени. *Потоком* здесь мы будем называть упорядоченную последовательность фрагментов данных (коими могут быть изображения или звуковые фрагменты). Вообще же у термина *поток* множество зависящих от контекста значений. Так, например, существует *потоковое* аудио или видео (и соответствующие форматы данных, большинство из которых, впрочем, являются именно *потоковыми*). Их особенностью является то, что фрагменты такого потока могут быть воспроизведены независимо от других элементов. Фильм или звук в *потоковом* формате, например, можно воспроизводить в процессе скачивания. Также аудио-видео потоки могут быть и вовсе бесконечными: интернет радио, IP-TV, IP-телефония, и т.д.

Если данные отдельных изображений уже занимают значительные объемы и требуют сжатия, то о видео данных и говорить нечего – сжатие просто необходимо. Особенности видео позволяют использовать тот факт, что подряд идущие кадры в большинстве случаев весьма похожи: остается неподвижным фон или большая часть фона сдвигается, не изменяясь (движение камеры). Таким образом, имеет смысл независимо кодировать первый кадр серии, а затем кодировать разность (отличия) между ним и последующим кадром. Конечно, время от времени придется вставлять в поток независимые кадры. Можно кодировать кадр на основе его отличий как от предшествующего ему кадра, так и от следующего за ним. Например, герой движется по экрану. Рассмотрим три подряд идущих кадра. На кадре 1 герой закрыл часть фона, который проступил на кадре 2, а на кадре 3 фон, скрытый на кадре 1 приоткрылся еще больше. Итак, кодируя кадр 2 на основе кадра 1 невозможно определить закрытый героем фон, но его можно взять из кадра 3. Поэтому кодер запишет сжатые кадры в поток в последовательности 1,3,2. Декодер же декодирует кадры 1,3 и по ним восстановит кадр 2. Способов кодирования разности между кадрами существует множество, и подробно обсуждать их мы не будем.

Отметим, что основные требования к скорости, конечно, предъявляются к декодеру видео. Процесс кодирования сопряжен с гораздо большими затратами на вычисления и память и происходит на порядки медленнее.

#### Формат AVI

В качестве примера формата видео данных, мы рассмотрим формат AVI. Аббревиатура AVI (Audio Video Interleaved) дословно означает «Чередование Аудио и Видео». Как и рассмотренный ранее аудио формат WAV, AVI основан на более универсальном *бинарном* формате RIFF (Resource Interchange File Format – формат файла для обмена ресурсами).

RIFF использует little-endian порядок байт. Основным строительным кусочком RIFF является chunk (фрагмент, кусок, блок и т.д. данных) имеющий следующую базовую структуру:



Рис. 2.68 Базовая структура блока RIFF

Поля имеют следующее назначение.

* chunkId – идентификатор блока. Содержит ASCII строку из четырех символов. Иногда значение трактуется как четырехбайтное целое, например, 'RIFF' (0x46464952), 'LIST' (0x5453494C). По значению данного поля определяют, как интерпретировать полезную нагрузку в поле данных chunkData.
* chunkSize – размер поля данных в байтах.
* chunkData – полезные данные в количестве chunkSize байт. Могут содержать вложенные chunk-блоки.

Chunk в файле идут подряд друг за другом, поэтому, зная размер chunkSize поля данных некоторого chunk всегда можно найти начало chunk, следующего за ним. В документации chunk для простоты часто записывают в текстовой форме, опуская размер: '<chunkId>'(<chunkData>). Например, запись: 'strh'(<Данные заголовка>) определяет chunk с идентификатором 'strh', содержащий «данные заголовка».

Далее для формата AVI определяется несколько специфичных chunk (обратите внимание, что все специфичные chunk на самом деле являются базовой структурой блока). Chunk заголовка файла имеет идентификатор chunkId='RIFF' и определяется так:



Рис. 2.69 Структура chunk заголовка RIFF AVI файла

Заголовок помимо стандартных полей в поле данных содержит идентификатор типа файла fileType (ASCII строка из четырех символов), равный для AVI файлов 'AVI ' (0x20495641). Поле fileData содержит вложенные chunk, размечающие полезные данные файла. В текстовой форме записывают так: 'RIFF'('AVI', <fileData>).

Еще одним специфичным chunk является список (chunkId='LIST'), это второй специфичный chunk который помимо заголовка файла (chunkId='RIFF') может содержать вложенные chunk.



Рис. 2.70 Структура chunk списка LIST, содержащего другие chunk AVI файла

Поле listType содержит значение идентификатора типа списка, состоящее из четырех символов ASCII. По этому значению определяется структура поля listData. Поле listData содержит вложенные chunk. В текстовой форме записывают так: 'LIST'('<listType>', <listData>).

Структура файла AVI приведена на Рис. 2.71 (пунктиром обозначены необязательные элементы). Мы ограничимся весьма поверхностным рассмотрением первой версии формата (активно используемой и по сей день).



Рис. 2.71 Формат файла AVI

Та же структура, но в текстовой форме (в квадратных скобках указаны необязательные элементы):

|  |
| --- |
| 'RIFF'('AVI '  'LIST'('hdrl'  'avih'(<Заголовок AVI файла>)  'LIST'('strl'  'strh'(<Заголовок потока 0>)  'strf'(<Формат потока 0>)  [ 'strd'(<Дополнительный заголовок потока 0>) ]  [ 'strn'(<Имя потока 0>) ]  ...  )  'LIST'('strl'  'strh'(<Заголовок потока 1>)  'strf'(<Формат потока 1>)  [ 'strd'(<Дополнительный заголовок потока 1>) ]  [ 'strn'(<Имя потока 1>) ]  ...  )  ...  )  'LIST'('movi'  '00dc'(<Поток 0. Сжатые видео-данные>)  '01wb'(<Поток 1. Аудио-данные>)  ...  )  [  'idx1'(  <Ссылки на все chunk потоков  относительно начала списка 'movi'>  )  ]  ) |

Опишем кратко chunk в составе файла. Мы не будем раскрывать подробную структуру каждого chunk, ограничившись для некоторых поверхностным описанием. Собственно файл содержит два обязательных списка hdrl (**h**ea**d**e**r** **l**ist) и movi (**movi**e). Необязательным chunk в файле является idx1.

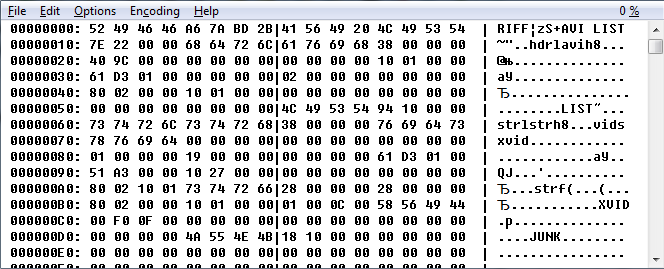


Рис. 2.72 AVI файл открыт в hex-редакторе

Список hdrl содержит chunk заголовка файла avih и произвольное количество списков strl, описывающих соответствующие *потоки*. Обычно *потоков* два: один – поток кадров изображения, второй – поток звука. Но, например, фильм может сопровождаться несколькими звуковыми потоками (по потоку на каждый перевод фильма на разные языки) или несколькими потоками изображений для стереоэффекта. А для немого кино можно ограничится и одним потоком.

Заголовок файла avih содержит общую информацию о файле целиком. В частности указывается количество *потоков* в файле, высоту и ширину видео в пикселях, количество микросекунд, в течение которых происходит смена кадров и т.д.

После заголовка в составе списка hdrl идут списки strl (**str**eam **l**ist), описывающие отдельный поток в файле. Список strl должен содержать следующие chunk: strh (**str**eam **h**eader) – заголовок потока и strf (**str**eam **f**ormat) – описание формата данных потока. Заголовок потока strh содержит общее описание потока: тип потока (аудио, изображений, текстовый, MIDI[[41]](#footnote-41)), его приоритет, размер элемента потока, отсчет времени, когда нужно начать воспроизводить поток и т.д. Формат потока strf должен идти следом за заголовком потока strh. Полезные данные в составе strf зависят от типа потока, например для потока изображений это структура, похожая на IMAGE\_HEADER (см. Рис. 2.46). Необязательными chunk являются, например, strd и strn. strd (additional **str**eam **d**ata), содержит дополнительную информацию о потоке, обычно формируемую кодером. Необязательный strn (**str**eam **n**ame) содержит ASCII строку с кратким текстовым описанием потока.

Описатели потоков strl, в составе списка hdrl ассоциированы с данными потоков в списке movi. Первому элементу strl в списке hdrl соответствуют данные нулевого потока, второму – данные первого потока, и т.д.

Список movi расположен следом за списком hdrl в файле. Список movi содержит фрагменты данных для определенных в hdrl потоков: кадры видео, звуковые фрагменты, MIDI, фрагменты текста. Собственно фрагменты содержатся в chunk, идентификаторы которых (chunkId) формируются следующим образом: первые два ASCII символа – десятичное представление номера потока, последние два ASCII символа дополнительно определяют тип данных фрагмента:

|  |  |
| --- | --- |
| chunkId[2…3] | Тип данных |
| 'db' | Несжатые кадры видео |
| 'dc' | Сжатые кадры видео |
| 'pc' | Изменение палитры |
| 'wb' | Звук |
| XX | Текст (произвольный chunkId[2…3]) |

Например, если поток 0 содержит звуковую дорожку, то соответствующие ему chunk в списке movi будут иметь chunkId='00wb'. Текстовые фрагменты могут иметь произвольный двухбуквенный код типа данных. Отдельно стоит сказать о chunk с кодом 'pc' – изменение палитры. Естественно в видео применяются индексирование цветов, чтобы уменьшить размер данных. Поэтому, если на сцене происходит значительная смена цветов, то требуется сформировать палитру заново.

Последним рассмотрим необязательный chunk с идентификатором 'idx1'. Он содержит упорядоченный массив структур (см. Рис. 2.73), в составе которых имеются смещения на соответствующие фрагменты данных в списке movi (см. Рис. 2.71) относительно его начала.



Рис. 2.73 Структуры в составе idx1

Назначение полей структуры следующее:

* chunkId – идентификатор chunk в списке movi, на который ссылаемся (например '00wb').
* flags – битовая маска, определяющая некоторые характеристики элемента, на который ссылаемся. Например, влияет ли он на ход времени просмотра (изменения палитры, например, не влияет)
* offset – смещение chunk, на который ссылаемся, в байтах относительно начала данных списка movi (для быстрого перехода).
* size – размер chunk, на который ссылаемся.

Проиндексировав таким способом список movi, разработчики формата упростили, например, переходы в процессе просмотра видео (теперь для этого уже не требуется просматривать список movi, достаточно в массиве idx1 быстро найти нужный отсчет времени и по смещению обратится к элементу потока для воспроизведения).

**Вопросы**

1. Каковы особенности сжатия видео?
2. Каковы особенности потокового видео?
3. Оцените объемы данных для одного часа видео (24 fps), с размером экрана пикселей, если RGB кадры не сжаты, и 32 битный двухканальный PCM звук (44 КГц) также не сжат.
4. Какие типы потоков могут быть представлены в AVI файле?
5. Как в формате AVI формируется значение chunkId сэмплов разных типов в блоке 'movi'?
6. Зачем в формате AVI введен блок 'idx1'?
7. Приведите общую структуру формата AVI, какие данные содержат отдельные элементы?

## Кодирование действий

*Программа* – это *данные*, в которых представлены *действия*. То, что выполняет *действия*, представленные *программой*, называется *интерпретатором*. Иногда говорят, что «программа выполняется» или «программа работает», вместо того, чтобы говорить «интерпретатор выполняет действия, представленные программой». Интерпретатором может быть *устройство* (например, процессор), другая *программа* (которая, выполняясь, будет выполнять целевую программу), или вовсе живое существо (например, человек).

При таком «широком» определении терминов *программа* и *интерпретатор* отдаляются от привычного контекста, в котором эти термины чаще всего употребляются, а именно от *программирования*. Допустим, у нас имеется некоторое изображение в векторном формате. Эти *данные* содержат инструкции для средства просмотра изображений наподобие таких: «отобразить окружность такого-то цвета, радиусом таким-то, с центром в точке такой-то, толщиной линии такой-то». Получается, что данные изображения это *программа*, а средство просмотра изображений – *интерпретатор* этой программы. Но закрадываются сомнения: изобразительное творчество отношения к программированию имеет весьма отдаленное. «Сужать» определение *программы* и *интерпретатора* мы не будем, мы сузим тему.

Сужая тему до программирования, отметим: *программой* представлены обычно не *любые* действия. Программой обычно (но не всегда) представлен *алгоритм* или *метод*. Дать однозначное и одновременно простое определение алгоритму вовсе не просто. И дано их немало.

Вот лаконичное определение: *алгоритм* – это точный набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное время.

Вот (сокращенное) определение от Дональда Кнута, автора фундаментального труда «Искусство программирования»: Алгоритм – это не просто набор конечного числа правил, задающих последовательность выполнения операций для решения задачи определенного типа. Помимо этого он имеет пять особенностей.

1. Конечность. Алгоритм всегда должен заканчиваться после выполнения конечного числа шагов. То, что не имеет данной особенности, но имеет все остальные, называется *методом*.
2. Определенность. Каждый шаг алгоритма должен быть точно (однозначно) определен.
3. Ввод. Алгоритм имеет некоторое (возможно равное нулю) число *входных данных*, т.е. величин, которые задаются до начала его выполнения или определяются динамически во время выполнения.
4. Вывод. У алгоритма есть одно или несколько *выходных данных*, т.е. величин, имеющих вполне определенную связь с *входными данными*.
5. Эффективность. Алгоритм считается эффективным, если все его операторы достаточно просты для того, чтобы их можно было выполнить в течение конечного промежутка времени.

Программы обычно задаются с помощью *языков программирования,* в которых каждое выражение имеет строго определенное значение. Примером такого языка может быть, например, язык машинных команд, программы на котором, интерпретируется непосредственно процессором (аппаратной частью). Программировать на этом языке весьма сложно хотя бы потому, что этот язык предназначен для выполнения автоматом, а не человеком. Программа на таком языке – это бинарные данные. Для человека удобнее язык ассемблера – текстовое представление машинных команд. Машинным командам однозначно соответствуют текстовые мнемоники – легко запоминаемые человеком обозначения команд. Язык ассемблера всем хорош, но время не стоит на месте, появляются все новые модели процессоров, более совершенные и порой несовместимые с предшественниками (не понимающими их язык команд), а с ними появляются и новые языки ассемблеров. На заре вычислительной техники это проявлялось весьма остро – приходилось часто переписывать существующие программы для новой аппаратуры. Решением стали языки высокого уровня, в которых отсутствует «привязка» к конкретным машинным командам. Их выражения весьма абстрактны и приближены к выражениям естественных (человеческих) языков.

Мы подошли к еще одному важному понятию: *транслятор*. Исходя из изложенного выше, очевидна необходимость перевода (трансляции) с языка высокого уровня или языка ассемблера в язык машинных команд, так как единственным «железным» интерпретатором является процессор. *Транслятор* переводит программу на исходном (входном) языке в *эквивалентную* ей программу на результирующем (выходном) языке. Чаще всего *транслятор* сам является программой. Теперь достаточно переписать единственную программу – *транслятор* с языка высокого уровня – на новую платформу, а затем все остальные программы, написанные на языке высокого уровня, перенести в автоматическом режиме с помощью *транслятора*. *Компилятором* называют специфичный транслятор, переводящий программу с языка высокого уровня только в язык машинных команд (или язык ассемблера). *Транслятором* в отношении данных (не программ) является *конвертор.* *Конвертором* называется программа, переводящая данные из одного формата в другой. Так, например, документ из формата doc можно конвертировать в формат pdf, а изображение из формата BMP – в формат TIFF.

Далее, на нескольких примерах мы постараемся дать представление о кодировании действий.

### Регулярные выражения: поиск элементов регулярных множеств

Мы с небольшой «натяжкой» будем считать *регулярные выражения языком программирования*, с помощью которого представлен (пусть весьма смутно) *алгоритм* определения принадлежности фрагмента *текста* некоторому регулярному множеству. Далее обо всем подробно.

В своё время регулярные выражения были прорывом в технике обработки *текстов*. И они не теряют своих позиций со временем. Регулярные выражения являются частью синтаксиса многих языков программирования, в остальных же языках регулярные выражения доступны через функции стандартных библиотек. Более того, так как основным инструментом программиста является текстовый редактор, то практически все достойные внимания среды разработки поддерживают поиск и замену на основе регулярных выражений. Начнем с небольшого теоретического введения.

Мы уже сталкивались с *множествами*, например, когда обсуждали принципы кодирования. *Множество* – это фундаментальное, неопределяемое понятие математики. Можно сказать, что *множество* – это любая определенная совокупность объектов. Объекты, из которых составлено множество, называются его *элементами*. *Элементы* множества различны и отличимы друг от друга. Если является элементом множества , то говорят что принадлежит множеству и сей факт обозначается . В противном случае говорят, что не принадлежит множеству и обозначается . Если множество небольшое, то его можно задать простым перечислением элементов. Обычно элементы заключают в фигурные скобки и разделяют запятыми. Например, , а . Когда множество достаточно велико, то его задают с помощью условия, определяющего принадлежность элемента множеству или задают алгоритм, порождающий элементы множества. Например, множество целых чисел от одного до тысячи можно задать так . Множество не содержащее ни одного элемента, называется пустым и обозначается . Объединением двух множеств и (обозначается ) называется множество, содержащее элементы обоих множеств: , где - соответствует союзу «или». Например, . Результат объединения нескольких множеств записывается так:

Также мы имели дело и с *цепочками* символов (словами) в алфавите . И знаем, что *префиксом*, *началом* или *приставкой* слова называется цепочка , если , а *постфиксом* или *окончанием* называется, соответственно, цепочка . Пустая цепочка, которую далее будем обозначать , это такая цепочка, что для любой цепочки справедливо .

Уже почти все готово, чтобы дать определение регулярного *множества*, а затем и *выражения*, но прежде приведем еще ряд специфичных обозначений. *Конкатенацией* множеств и , содержащих цепочки, будет множество , где - соответствует связке (союзу) «или». Например, конкатенацией множеств и будет множество из шести элементов . Возведением в степень () множества , обозначаемое как , будем называть конкатенацию , причем . Например,

*Регулярное множество* в алфавите символов рекурсивно определяются так.

1. - пустое множество является *регулярным множеством* в алфавите символов .
2. – множество, содержащее пустую цепочку, является *регулярным множеством* в .
3. – множество, содержащее один символ алфавита (), является *регулярным множеством* в .
4. Если и – *регулярные множества* в алфавите , то *регулярными* в также являются множества.
   1. . Объединение множеств.
   2. . Конкатенация.
5. Если – *регулярное множество* в , то (итерация ) является *регулярным множеством* в .
6. Ничто другое не является *регулярным множеством* в алфавите .

Отдельного упоминания заслуживает итерация множества – бесконечное множество , являющееся объединением всех степеней (вплоть до бесконечности) регулярного множества . Например, итерацией множества является множество .

Регулярные множества находят множество применений на практике. Например, записи чисел, адресов электронной почты (e-mail), URL, имен переменных и констант в языках программирования, и т.д. являются элементами регулярных множеств.

*Регулярное множество* можно задать с помощью *регулярного выражения*. Регулярное выражение в алфавите T и обозначаемое им множество рекурсивно определяются так:

1. – *регулярное выражение,* обозначающее *регулярное множество* .
2. – *регулярное выражение,* обозначающее *регулярное множество* .
3. – *регулярное выражение* ()*,* обозначающее *регулярное множество* .
4. Если и – регулярные выражения, обозначающие соответственно множества и то.
   1. – *регулярное выражение,* обозначающее *регулярное множество* . Объединение.
   2. – *регулярное выражение,* обозначающее *регулярное множество* . Конкатенация.
5. Если – *регулярное выражение*, обозначающее *регулярное множество* то, - *регулярное выражение*, обозначающее *регулярное множество* . Итерация.
6. Ничто другое не является *регулярным выражением* в алфавите .

Операция *итерации* имеет наивысший приоритет, затем идет операция *конкатенации*, затем – *объединения*. Поэтому, как и в арифметике, лишние скобки порой (на усмотрение) не пишут: . Очень часто вместо пишут .

Например, выражение порождает регулярное множество . Регулярное выражение определяет представление восьмеричных чисел в языке программирования java см. параграф 2.4.

Так в теории формальных языков *формально* определяются регулярные выражения. Па практике же, с целью упростить процесс описания регулярных множеств, используют некоторые *расширения* формально определенных регулярных выражений. Такими расширениями являются, например, регулярные выражения POSIX (**P**ortable **O**perating **S**ystem **I**nterface for Uni**x** – переносимый интерфейс операционных систем Unix) или PCRE (Perl Compatible Regular Expressions - регулярные выражения, совместимые с используемыми в языке программирования Perl).

Мы рассмотрим основные элементы PCRE, ставшие основой практически для всех спецификаций языков программирования и библиотек. Итак, в составе PCRE выделяют:

* одиночные символы;
* классы символов;
* альтернативы;
* квантификаторы;
* мнимые символы;
* ссылки на найденный текст;
* и т.д.

Любой *одиночный символ* в составе PCRE соответствует сам себе, если только он не является служебным символом (спецсимволом), играющим особую роль. Служебными символами являются «\», «|», «(», «)», «[», «]», «{», «}», «\*», «+», «^», «$», «?» и «.». Роль этих символов станет ясна в дальнейшем, сейчас же отметим, что если необходимо, чтобы служебный символ соответствовал самому себе, то перед ним нужно поставить символ «\». То есть регулярное выражение, производящее поиск символа «\» в тексте будет выглядеть так «\\». Кроме того, в PCRE вводятся ряд специальных обозначений, соответствующих нескольким символам или символам, ввод которых затруднен с клавиатуры. Например:

* «\d» – соответствует одной десятичной цифре;
* «\D» – соответствует любому символу, *кроме* десятичной цифры;
* «\s» – соответствует любому из символов-разделителей (пробельных символов) слов (пробел, табуляция, перевод строки и т.д.);
* «\S» – соответствует любому символу, за исключением пробельного;
* «\w» – соответствует алфавитно-цифровому символу (любой латинской букве, десятичной цифре и знаку подчеркивания «\_») ;
* «\W» – соответствует любому символу, за исключением алфавитно-цифрового;
* «\r» – символ перевода каретки <CR>;
* «\n» – символ новой строки <LF>;
* «\R» – не зависимый от платформы (операционной системы) символ-разделитель строк;
* «\xHH» – соответствие символу ASCII с кодом из двух шестнадцатеричных (HH) цифр;
* «\uHHHH» – соответствие символу Unicode с кодом из четырех шестнадцатеричных (HHHH) цифр.

Кроме того, спецсимвол точка «.» соответствует *любому* символу, за исключением символа-разделителя строк.

*Классы символов* определяют соответствие одному символу из множества. Класс – это список символов, заключенный в квадратные скобки («[», «]» – спецсимволы). Причем можно указать как отельные символы, так и диапазон. Диапазон задается двумя крайними символами диапазона, разделенными тире. Если требуется указать тире, то перед ним ставится символ «\». Например «[abcde]» - любая из букв «а», «b», «c», «d», «e». Это же выражение можно записать короче, так как символы идут по алфавиту: «[a-e]». Если нужно например определить класс из знака «-» и всех цифр, то можно это сделать так: «[\-0123456789]», «[\-0-9]» или, например, «[\-\d]». Если сразу после открывающей квадратной скобки следует спецсимвол «^», то будет выполняться поиск соответствия символу, не входящему в класс, то есть смысл выражения меняется на противоположный: например, «[^a-e]» - *любой символ*, кроме: «а», «b», «c», «d», «e».

*Альтернативы* – соответствуют операции *объединения* в формальном определении регулярных выражений. Альтернативные *регулярные выражения* разделяются спецсимволом «|» и обычно заключаются в круглые скобки. Например, регулярному выражению «(саша|паша)\+(маша|даша)=Л» соответствует множество из четырех возможных вариантов резьбы по дереву.

Квантификаторы ставятся после регулярного выражения и определяют количество повторений шаблона. Выделяют следующие квантификаторы:

* «\*» – *ноль* или несколько повторов (итерация в формальном определении);
* «+» – *один* или несколько повторов;
* «?» – ноль или один повтор;
* «{n}» – ровно n повторов;
* «{n,}» – по крайней мере n повторов;
* «{n,m}» – от n до m повторов.

Про «+» и «\*» все известно из формального определения. Остальное лишь более удобная запись:

Отдельно стоит сказать, что по умолчанию квантификаторы «\*», «+», «?», «{n,}», «{n,m}» являются «*жадными*» (greedy). То есть они выделят по возможности самый длинный фрагмент из всех возможных. Например, при поиске в тексте «aaaaaaaaaa» шаблону «a\*a» будет соответствовать весь текст. Сделать квантификаторы «ленивыми» (lazy) можно, поставив после них знак вопроса «?». Тогда в приведенном примере для «ленивого» варианта «a\*?a» результатом поиска будет одна бука «a».

*Мнимые символы* не соответствуют символам текста! Они соответствуют выполнению определенного условия (assertion), например:

* «^» – начало строки текста;
* «$» – конец строки или позиция перед символом начала новой строки, расположенного в конце;
* «\b» – граница слова;
* «\b» – отсутствие границы слова.

Иногда нужно сослаться на подстроку текста, для которой получено совпадение с некоторой частью регулярного выражения. Это можно сделать! Для этого необходимую часть следует заключить в круглые скобки (что конечно не меняет смысл выражения). Каждому выражению, заключенному в скобки будет соответствовать переменная с определенным номером. Чтобы к ней обратится необходимо перед её номером поставить «\». Например, если вы желаете найти целое положительное число, являющееся значением элемента XML (элемент без атрибутов, см. параграф 2.7.2), то вы можете задать такое выражение: «<(\w+)>\d+</**\1**>». В переменной \1 будет содержаться текст, который был найден как соответствующий регулярному выражению «(\w+)».

На фрагменты текста после поиска можно сослаться и «извне» регулярного выражения. Причем независимо от того, в каком виде вы использовали регулярные выражения: при работе с хорошим текстовым редактором, или в создаваемой программе. Условно можно считать, что после того, как поиск был успешно завершен, «на выходе» получается массив, содержащий значения соответствующих переменных. В элементе с индексом 0 содержится текст соответствующий шаблону целиком, в элементе с индексом 1 содержится значение, соответствующее первой части регулярного выражения, в элементе с индексом 2 – второй части и т.д. Так как части регулярного выражения заключенные в скобки могут быть вложенными, то нумеруются они по открывающим скобкам, слева направо. Мы будем обращаться к значениям элементов массива так: $n, где n – индекс элемента массива (во многих редакторах обращение происходит именно так). Например, если задано регулярное выражение «(\w(\w(\w)))((\w)(\w))» и текст «ABCDE», то значения переменных будут: $0=«ABCDE», $1=«ABC», $2=«BC», $3=«C», $4=«DE», $5=«D», $6=«E». С помощью ссылок на фрагменты текста, соответствующего шаблону регулярного выражения очень удобно выполнять самые нетривиальные замены в тексте. Например, переставить байты в 32 битном шестнадцатеричном числе в обратном порядке можно так: заменить текст, соответствующий регулярному выражению «0x([0-9a-fA-F]{2})([0-9a-fA-F]{2})([0-9a-fA-F]{2})([0-9a-fA-F]{2})» на текст «0x$4$3$2$1». Число «0x01ABCDEF» будет заменено на «0xEFCDAB01».

Например, редактор кросс платформенной среды разработки Eclipse позволяет программисту использовать всю мощь PCRE в процессе написания программы:

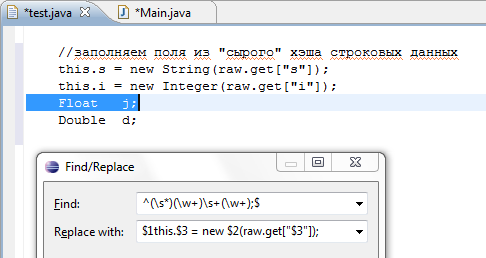


Рис. 2.74 Регулярные выражения в среде Eclipse за работой

Регулярные выражения позволяют программисту уйти от написания собственных алгоритмов для решения рутинных задач специфичного поиска или замены в тексте, да и обычному пользователю они могут стать неплохим подспорьем при работе с данными в текстовом формате.

**Вопросы и задания**

1. Какое множество порождает регулярное выражение ( – пустая цепочка):

|  |  |
| --- | --- |
| Вариант | Регулярное выражение |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |

1. Сколько элементов в регулярном множестве, соответствующем регулярному PCRE выражению:

|  |  |
| --- | --- |
| Вариант | Регулярное выражение |
| 1 | \d{3,4}[a-z]{2} |
| 2 | 0x[0-9a-fA-F]{1,8} |
| 3 | [0-7]{1,8}|(раз|два|три)(четыре|пять) |
| 4 | \d?\d?\d? |
| 5 | ^\b[a-f]{1,4}\d?\b$ |
| 6 | 0[0-7]{1,7}? |

1. Организуйте в тексте поиск чисел. Примеры чисел: 0, -5, 0.1, -2.24, -2.123e-15, -31.123E+7. Запись числа -31.123E+7 называется научной нотацией и её следует понимать так: .
2. Организуйте поиск адресов e-mail в тексте. Примеры адресов: [kafevm@mail.ru](mailto:kafevm@mail.ru), [e.mail@dot.net.com](mailto:e.mail@dot.net.com), и т.д.
3. Организуйте поиск URL http протокола в тексте. Примеры: <http://my.ru/index.php?a=b&c=d>, <http://your.my.ru/a/b/c/index.html>, <http://your.my.ru/a/b/1/>.
4. Вам необходимо найти все места в статье на 20 страниц, где упоминается Николай Васильевич Гоголь. Вам не хочется пропустить ни одного упоминания. Попробуйте составить подходящее регулярное выражение, способное вам помочь. Пусть будут ложные срабатывания, ничего страшного. Постарайтесь учесть падежи и сокращения.
5. Даты у разных народов пишутся по-разному. Например, в России принято писать сначала день, потом месяц, потом год: «29.02.1982», «29/02/1982». В США, например, даты пишут в таком порядке: месяц, день, год. Выполните замены американских дат российскими, учтя, что разделителями могут быть точка, дефис и косая черта. Числа, соответствующие дням и месяцам не обязательно дополняются ведущим нулем: «1.1.2011» В тексте не встретится сокращенных записей для года (иногда пишут 82 вместо 1982).
6. В базе данных скопилось множество номеров телефонов одного города. Пусть это будет Киров с международным кодом 8(8332). В свое время под номер была отведена строка, и операторы вводили номера телефонов, сообразуясь со своими эстетическими пристрастиями: 8-8332-123456, 8 8332 12 34 56, 8-8332-123-456, 8-8332-12-34-56, 8(8332)12 34 56. И даже без кода: 12 34 56, 123-456. Пора положить конец хаосу. Да будет единый: 8(8332)XX-XX-XX.

### Операции, вычисления, программы

Понятие *алгоритм* было рассмотрено весьма подробно, и уже известно, что алгоритм выполняется пошагово, инструкция за инструкцией, операция за операцией. А значит и программа состоит из дискретных элементов – операторов. *Операторами* будем далее называть *выражения* конкретного *языка программирования* для представления *операций* в программе.

Операторы подразделяют по различным признакам.

1. По назначению (*арифметические*, *логические*, *присваивания*, условные, циклов и т.д.).
2. По значению *приоритета* относительно других операторов.
3. По количеству связываемых аргументов (*одноместные*, *двуместные*, трех и N-местные).
4. По отношению к аргументам (*левоассоциативные*, *правоассоциативные*).
5. И так далее…

Первые два пункта не вызывают затруднений. То, что операция умножения выполняется раньше операции сложения, известно еще со школьной скамьи. И все языки программирования стараются придерживаться этих нехитрых правил. Так что на знаменитый вопрос с подвохом: «сколько будет два плюс два умножить на два?» – многие интерпретаторы языков программирования ответят правильно: 6. Но на некоторых языках привычная запись этого простого математического выражения будет выглядеть совсем иначе. Выражение «2+2\*2» на языке ЛИСП будет выглядеть так: «((+ 2 (\* 2 2)))», а в обратной польской записи надо будет записать его так: «2 2 2 \* +». Терпение, скоро все станет ясно.

Программа практически на любом языке программирования представляет собой одну большую строку текста. То есть практически все программы представлены в *текстовом формате*. Да, программисты живут в одномерном мире. Программа может быть создана в простейшем текстовом редакторе. Поэтому, почти у каждого (конечно, но не у первого и не у последнего) символа есть два соседа, – справа и слева соответственно. Именно поэтому *одноместные* и *двуместные* операции в отличие от трех и более местных очень удобно обозначить *одним* символом. Большинство арифметических операций – одноместные и двуместные. Приведем примеры из конкретных языков.

* «2+3». Сложение – двуместная операция: аргумент слева («2» – первый) связан символом операции «+» с аргументом справа («3» – вторым) и под записанным выражением следует понимать результат 5. Большинство языков: C/C++, C#, Java, Pascal, Basic, Ada,… проще назвать исключения
* «! true». Логическое отрицание (из языка C/C++ и ему подобных) – операция одноместная. Символ операции «!» связывает логический аргумент справа «true» (ИСТИНА) в один результат false (ложь).
* «A’». Транспонирование матрицы (из языка среды MATLAB). Символ одноместной операции «’» связывает аргумент «A» (имя матрицы) слева, в результате чего получим транспонированную матрицу «A».
* «X ? 2 : 3». Оператор языка C/C++. Трехместный. Если первый аргумент («X») – истина, возвращается второй аргумент, если нет – третий. При X=true возвращает результат 2, при X=false возвращает 3.

Ассоциативность поясним на примерах одноместных и двуместных операций.

* Для одноместных операций при записи: «» решить «» относится к «» (тогда операция правоассоциативна – «») или к «» (тогда левоассоциативна «»). Примеры одноместных левоассоциативных операторов: унарный минус («-2» – общематематический), логическое отрицание («! true» – C/C++), прединкремент («++i») и преддекремент («--i») в C/C++. Примеры одноместных правоассоциативных: факториал («10!» – общематематический), транспонирование матрицы в MATLAB («A’»), постинкремент («i++») и постдекремент («i—«) в C/C++.
* Для двуместных операций при записи «» следует решить, что имеется в виду: «» (левоассоциативность) или «» (правоассоциативность). Например, операция вычитания – левоассоциативная, а присваивания – правоассоциативная.

Современному программисту приходится писать программы на *языках программирования высокого уровня*. *Языки высокого уровня* приближены к естественным языкам или предоставляют средства максимально компактного описания решения задач. Тем не менее, выполнять программу должен интерпретатор. Единственным «настоящим» интерпретатором вычислительной системы является процессор. Язык процессора прост и однозначен. Этой простоты требует аппаратная реализация. В частности команды машинного языка исполняются процессором последовательно, друг за другом, время от времени специальные команды перехода заставляют процессор пропустить группу команд или перейти на несколько команд назад. Человек же, записав программу «2+2\*2» отнюдь не последователен в вычислениях, так как в свое время придумал *приоритет* операций.

Поэтому возникает необходимость в программах особого рода: *трансляторах*. Одна из фаз преобразования ЯВУ программы в машинный код называется *синтаксический анализ*. В результате выполнения этой фазы, будет построено *дерево синтаксического разбора*. Строгое определение понятию «дерево» дано не будет. Дерево – это корень ветви и листья, это иерархический граф без циклов. Иллюстрации внесут ясность. В приведенных ниже примерах из каждой вершины графа исходят всего две дуги по причине двуместности рассматриваемых операций, а сама операция «вписана» в вершину. Пример дерева разбора для «2+2\*2» приведен на Рис. 2.75.



Рис. 2.75 Пример дерева синтаксического разбора для выражений «2+2\*2» и «(2+2)\*2»

Не будут приведены также формальные методы построения деревьев синтаксического разбора, полагаясь на то, что читатель интуитивно сможет построить подобные деревья для известных ему языков. Представление выражения в виде дерева синтаксического разбора полностью однозначно. Например, для выражения «» мы получим два варианта дерева синтаксического разбора, если «» операция правоассоциативна (Рис. 2.76.а) или левоассоциативна (Рис. 2.76.б):



Рис. 2.76 Варианты деревьев разбора для «»

В зависимости от того как выполнен рекурсивный обход дерева будет получена *префиксная*, *инфиксная* или *постфиксная* запись выражения. Мы будем исходить из того, что все операции арифметические и двуместные. Когда суть станет понятна, при желании алгоритмы обхода можно будет модифицировать для других операций.

Начнем с *постфиксной* записи. *Постфиксную* запись получим в результате выполнения следующего рекурсивного алгоритма обхода дерева:

* Ввод: дерево
* Вывод: постфиксная запись для дерева
* Шаги:

1. Если дерево – это лист дерева (то есть состоит из одной вершины), то установить вывод в строку «» и завершить алгоритм. Здесь обозначение в вершине дерева .
2. Иначе рекурсивно получить вывод данного алгоритма для левого () и правого () поддеревьев, соответственно и . Вывод установить в «». Здесь обозначение в вершине дерева .

Разберем выполнение алгоритма на примере. Пусть на вход алгоритма поступает дерево :



Выполняя второй шаг, приходим к тому, что нужно выполнить алгоритм для поддеревьев и :



Для дерева получаем результат, выполняя первый шаг алгоритма: «1». Для дерева придется выполнить еще один рекурсивный спуск, разбив его на два поддерева и :



Для этих деревьев также получаются результаты при выполнении первого шага: «2» и «3» соответственно:



Вывод для дерева теперь можно установить в «2 3 \*», а вывод для исходного дерева в «1 2 3 \* +»:



Постфиксная запись называется также *обратной* *польской записью*. Она не содержит неоднозначностей, и в записи формулы в этой форме скобки отсутствуют.

Алгоритмы для получения *префиксной* и *инфиксной* формы отличаются от приведенного алгоритма для *постфиксной* записи только шагом 2 (точнее даже лишь формированием вывода на этом шаге).

Для *префиксной* формы шаг 2 будет таким: «Иначе рекурсивно получить вывод данного алгоритма для левого () и правого () поддеревьев, соответственно и . Вывод установить в «». Здесь обозначение в вершине дерева ». Узнаете запись на языке ЛИСП? Впрочем, можно обойтись и без скобок…

Для *инфиксной* формы шаг 2 будет таким: «Иначе рекурсивно получить вывод данного алгоритма для левого () и правого () поддеревьев, соответственно и . Вывод установить в «». Здесь обозначение в вершине дерева ». Скобки добавлены, чтобы устранить неоднозначность.

Вывод приведенных выше процедур представлен на Рис. 2.77. Стрелка вниз обозначает вход в рекурсию, а стрелка вверх – выход. Инфиксная форма не является однозначной, поэтому пришлось группировать операнды скобками (постройте дерево для выражения «(2+2)\*2» и получите инфиксное представление без скобок. Результат 2+2\*2 будет неверным).



Рис. 2.77 Обход дерева и полученные префиксная, инфиксная и постфиксная формы записи выражений

Сосредоточимся на постфиксной записи (её часто называют обратной польской записью). Как *вычислять* значение такого выражения? Продвигаясь слева направо по выражению доходим до первого символа операции. В зависимости от его арности выбираем слева от символа необходимое число аргументов и выполняем над ними соответствующую операцию. Все выражение заменяем результатом и продолжаем движение, повторяя изложенные действия: «2 2 2 \* +», «2 4 +», «6».

Мы переходим к построению модели *интерпретатора*, вычисляющего значения арифметических выражений. Вычислять значения постфиксной записи удобно с помощью интерпретатора – *стековой машины*. *Стек*, как известно это магазинная память, доступ к которой осуществляется с помощью двух команд.

* push(X) – поместить значение X на вершину стека.
* pop() – изьять с вершины стека значение и вернуть его. Y=pop() – в Y получим значение с вершины стека.

Стек работает по принципу последним пришел первым ушел (LIFO - Last In First Out). То есть элементы, помещенные в стек командами push будут извлечены в обратном порядке командами pop.

Алгоритм работы стековой машины для интерпретации постфиксной записи будет таким.

* Ввод: программа в виде выражения в постфиксной (обратной польской) записи
* Вывод: результат вычисления выражения
* Шаги:

1. Начать просмотр выражения слева направо.
2. Встретив число или идентификатор переменной (обозначим и то и другое X), поместить значение в стек: push(X), продвинуться на символ влево и перейти на шаг 2. Иначе к следующему шагу.
3. Если текущий символ выражения – символ операции, то изъять из стека командами pop() необходимое количество аргументов, выполнить над ними соответствующую операцию и поместить результат R в стек командой push(R). Продвинуться на символ влево и перейти на шаг 2. Иначе к следующему шагу.
4. Если достигнут конец выражения (программы), то на вершине стека получен результат: R=pop(), завершаем выполнение алгоритма, установив вывод в значение R, иначе перейти на шаг 2.

Работа стековой машины проиллюстрирована на рисунке Рис. 2.78.



Рис. 2.78 Интерпретатор «стековая машина» за работой

Кстати, популярная кроссплатформенная (от компьютеров до мобильных телефонов и чип-карт) виртуальная машина Java является стековой. Напоследок приведем более сложный пример, поясняющий, что стековая машина годится на большее, если дополнить её еще и командами условных и безусловных переходов.

Итак, пусть дано выражение на языке Pascal, позволяющее найти максимум среди двух чисел A и B:

|  |
| --- |
| **IF (A>B) THEN MAXI:=A ELSE MAXI:=B;** |

Данный оператор проверяет условие A>B, и если оно истинное, то переменной MAXI присваивается значение переменной A и происходит переход к следующему оператору, ежели условие A>B ложно, то переменной MAXI присваивается значение переменной B и выполняется переход к следующему оператору. Блок-схема алгоритма, соответствующая данному выражению приведена на Рис. 2.79.



Рис. 2.79 Блок-схема алгоритма выражения «IF (A>B) THEN MAXI:=A ELSE MAXI:=B;»

Дерево синтаксического разбора данной конструкции представлено на Рис. 2.79.



Рис. . Дерево разбора выражения «IF (A>B) THEN MAXI:=A ELSE MAXI:=B;»

По дереву синтаксического разбора была сформирована программа для стековой машины, система команд которой приводится далее. То есть выполнена *трансляция* программы на языке Pascal в команды стековой машины. Обход для вершины IF\_ELSE и формирование польской записи для нее несколько специфичен: нужно вычислить значение условия condition и, если оно истинно, выполнить переход на вычисление операторов true, а если оно ложно, выполнить переход на вычисление операторов false. При этом необходимо поставить между операторами true и false безусловный переход, чтобы «перепрыгнуть» ненужные операторы false, после выполнения операторов true.

Особенностью стековой машины в данном примере является то, что используется общая память как для хранения программы, так и для организации стека и хранения данных. Архитектура вычислительной машины, в которой память данных и память программ разделены, называется Гарвардской архитектурой. Пример интерпретации программы для A=3 и B=2 приведен на Рис. 2.80 и Рис. 2.81. Справа от ячеек приведены их адреса. Переменные A и B находятся по адресам 20 и 21 соответственно. Запись &X обозначает адрес X (например, &A=20). Результат должен получиться в переменной MAXI, то есть по адресу 22.

Приведем пояснения к введенным командам стековой машины. При этом мы обозначаем: [X] – значение по адресу X; [[X]] – значение по адресу, который берется как значение по адресу X (косвенная адресация). IP хранит адрес текущей инструкции, SP – адрес вершины стека.

* «PUSHVAL» – поместить в стек значение по адресу в ячейке памяти. [SP]:= [[IP+1]]; SP:=SP-1; IP:=IP+2.
* «PUSHADR» – поместить в стек значение в ячейке памяти. [SP]:=[IP+1]; IP:=IP+2; SP:=SP-1.
* «JMP» – безусловный переход на ячейку по адресу. IP:=[IP+1].
* «JMPFALSE» – переход на ячейку по адресу [IP+1] (IP:=[IP+1]), если на вершине стека находится значение ложь. Иначе IP:=IP+2. И в любом случае очистка стека: SP:=SP+1.
* «GR(>=)» – если [SP+1]>[SP] то в ячейку [SP+1] поместить true, иначе – false. SP:=SP+1.
* «:=» – скопировать значение в [SP] по адресу в [SP+1]: [[SP+1]]:=[SP]; SP:=SP+2.
* «HLT» – команда останова машины.



Рис. 2.81 Интерпретация транслированной программы «IF (A>B) THEN MAXI:=A ELSE MAXI:=B;»



Рис. 2.82 Интерпретация транслированной программы «IF (A>B) THEN MAXI:=A ELSE MAXI:=B;»

Мы рассмотрели примеры программ и интерпретаторов, которые производили *вычисления*. Но конечно, спектр программ гораздо шире: они оперируют не только с числами, но и символами текста, строками и т.д. а также со сложными структурами данных и объектами (структурами данных с поведением), которые современные языки программирования высокого уровня позволяют определять программисту.

**Задания**

1. Постройте дерево синтаксического разбора для выражения в инфиксной форме:

|  |  |
| --- | --- |
| Вариант | Выражение |
| 1 | (a+b/c)\*(d-e\*f) |
| 2 | (a+b-c)\*(d+e)\*f |
| 3 | (a+b-c)\*(d+e)-f |
| 4 | (a+b/c)-(d-e)\*f |
| 5 | (a+b)/c-d/e\*f |
| 6 | (a+b)/(c-(d-e)\*f) |

1. Переведите выражения из инфиксной формы в постфиксную (вариант возьмите из предыдущего задания).
2. Рассчитайте значения выражений, заданных в постфиксной форме:

|  |  |  |
| --- | --- | --- |
| Вариант | Постфиксная запись | Параметры |
| 1 | abc+d\*- | a=1,b=2,c=3,d=15 |
| 2 | ab+c-d\* | a=2,b=1,c=15,d=1 |
| 3 | ab\*cd+e-+ | a=1,b=7,c=3,d=11,e=2 |
| 4 | ab+cde-+\* | a=5,b=3,c=1,d=3,e=2 |
| 5 | abc-\*de+- | a=7,b=8,c=3,d=1,e=9 |
| 6 | abc/\*de+- | a=7,b=9,c=3,d=1,e=9 |

1. Преобразуйте в инфиксную форму выражение в постфиксной форме (вариант возьмите из предыдущего пункта).

### Языки программирования.

Как уже было сказано, программа представляет собой текст на некотором *языке программирования*. *Язык программирования* может быть определен с помощью описания того, как должна выглядеть программа (*синтаксис* языка) и что она означает (*семантика* языка). Описание семантики языка (например, как будет выполнено то или иное выражение) задача более сложная, чем описание синтаксиса.

Формально термин *язык* (просто язык, а не язык программирования) определяется так: *язык* в алфавите – это некоторое *множество цепочек* в алфавите . В параграфе 2.9.1, например, мы определили язык регулярных выражений (не сказав о том явно). Видно, что здесь отсутствует *семантика*, определяется только *синтаксис*. *Язык* – это подмножество множества .

Синтаксис подавляющего большинства языков программирования определяется с помощью *контекстно-свободной грамматики* или, как её еще называют, формы Бэкуса-Наура (BNF – Backus-Naur Form). BNF имеет четыре компонента.

1. Множество терминальных символов. Их еще называют терминалами или токенами. Это элементы множества . Это элементы, непосредственно составляющие *программу*.
2. Множество *нетерминальных символов* (*нетерминалов*). Нетерминальные символы обозначают последовательности токенов.
3. Множество продукций, каждая из которых состоит из *нетерминала*, называемого левой частью продукции, двоеточия и последовательности токенов и/или нетерминалов, называемой правой частью продукции.
4. Указание одного из *нетерминалных* символов как *стартового* или *начального*.

Грамматика определяется перечислением её продукций, причем первая в списке указывает стартовый символ. Для удобства записи правые части продукций, имеющих одинаковую левую часть (одинаковый нетерминал), могут быть сгруппированы с помощью символа «|». Форма Бэкуса-Наура является более мощным инструментом для описания языков, чем, например, регулярные выражения. Регулярные множества можно определить с помощью BNF. Уже упоминавшиеся деревья синтаксического разбора наглядно показывают, как стартовый символ порождает цепочку (строку) языка. Например, продукции A:BCD соответствует дерево:

.

Причем порядок следования слева направо потомков (BCD) узла (A) имеет значение. Поэтому для контекстно-свободной грамматики дерево разбора представляет собой дерево со следующими свойствами.

1. Корень дерева помечен стартовым символом.
2. Каждый лист помечен либо терминалом (токеном), либо пустой цепочкой .
3. Каждый внутренний узел представляет нетерминальный символ.
4. Если A является нетерминалом и помечает некоторый внутренний узел, а – отметки дочерних узлов, перечисленные слева-направо, то – продукция. Здесь - терминальные и/или нетерминальные символы.

Приведем пример описания грамматикой простых арифметических выражений в инфиксной форме (привычной человеку) в составе которых могут быть целые десятичные числа. Для удобства терминалы будем писать в кавычках, пробелы несущественны (если нужно задать пробел, мы запишем его так: « »). Целое число без знака можно определить так:

|  |
| --- |
| число : цифра | число цифра  цифра : «0» | «1» | «2» | «3» | «4» | «5» | «6» | «7» | «8» | «9» |

Видно, что этой грамматике могут соответствовать цепочки вроде «000», «01» и т.д., что не совсем красиво, но исправить ситуацию довольно просто:

|  |
| --- |
| число : цифра | не\_ноль список\_цифр  не\_ноль : «1» | «2» | «3» | «4» | «5» | «6» | «7» | «8» | «9»  список\_цифр : цифра | список\_цифр цифра  цифра : «0» | не\_ноль |

Далее опишем простые арифметические выражения:

|  |
| --- |
| выражение :  число |  выражение «+» выражение |  выражение «–» выражение |  выражение «\*» выражение |  выражение «/» выражение |  «(» выражение «)» |

Сведем в одну грамматику, переписав более кратко, для того, чтобы не загромождать дерево для грядущего примера.

|  |
| --- |
| в : ч | в «+» в | в «–» в | в «\*» в | в «/» в | «(» в «)»  ч : ц | нн сц  нн : «1» | «2» | «3» | «4» | «5» | «6» | «7» | «8» | «9»  сц : ц | сц ц  ц : «0» | нн |

Примеру программы «105/(2+3)» соответствует дерево на Рис. 2.83.



Рис. 2.83 Дерево разбора выражения «105/(2+3)»

Не будет обсуждаться то, что приведенная грамматика неоднозначна и не учитывает, например приоритет операторов (то есть одной и той же грамматике можно поставить в соответствие несколько деревьев разбора). Эта проблема решается либо заданием однозначной грамматики (отчего она становится гораздо менее наглядной), либо дополнительными средствами, задающими приоритет операторов.

Напоследок определим несколько синтаксических конструкций для операторов, использующихся практически во всех языках программирования, и зададим их семантику.

**Последовательность операторов, разделенных точкой с запятой.** *Синтаксис:*

|  |
| --- |
| операторы : оператор | операторы «**;**» оператор |

*Семантика*: операторы выполняются в порядке их перечисления слева направо.



**Условный оператор** (полная и сокращенная форма). *Синтаксис*:

|  |
| --- |
| оператор :  «**if**» «**(**» условие «**)**» «**{**» операторы1 «**}**» «**else**» «**{**» операторы2 «**}**» |  «**if**» «**(**» условие «**)**» «**{**» операторы «**}**» |

*Семантика*:



**Оператор цикла** (с предусловием и постусловием). *Синтаксис*:

|  |
| --- |
| оператор :  «**while**» «**(**» условие «**)**» «**{**» операторы «**}**» |  «**do**» «**{**» операторы «**}**» **while** «**(**» условие «**)**» |

*Семантика*:



То есть данный оператор выполняет операторы до тех пор, пока условие не станет ложным. В случае предусловия вначале проверяется условие и только если оно истинно выполнятся операторы. В случае постусловия вначале выполняются операторы, а затем выполняется проверка условия.

Конечно, в любом языке программирования существует множество других синтаксических конструкций с соответствующей семантикой. Например, программист может определять *функции*, то есть объединять набор операторов в один логический блок; блоку дается имя и определяются исходные данные (аргументы), необходимые для корректной работы; затем эти операторы можно выполнять лишь указанием имени функции и значений аргументов. Конечно, есть конструкции позволяющие определять переменные и их типы. Есть конструкции позволяющие определять собственные типы в виде композитных структур базовых типов. Многие современные языки программирования являются объектно-ориентированными. Синтаксические конструкции таких языков позволяют описывать классы: типы данных объявляемые программистом, которые представляют собой не только композитные наборы переменных, но и обладают определенным поведением (т.е. к ним логически «привязаны» еще и функции). Но эта тема уже выходит за рамки нашего обсуждения.

Отдельно следует сказать о трансляторах, позволяющих создавать трансляторы и компиляторы для новых (или весьма специфичных) языков программирования. Компиляторы компиляторов. В исходных текстах программ для таких трансляторов определяется синтаксис нового языка программирования, алгоритмы построения деревьев разбора выражений нового языка, способы их трансляции в программы на целевом языке. Примерами являются трансляторы Yacc, Bison, Byacc, позволяющие описывать синтаксис нового языка программирования в форме BNF. В результате получается исходный код компилятора на языке C/C++/Java.

**Вопросы**

1. Что такое форма Бэкуса-Наура? Для чего она предназначена?

**Задания**

1. Постройте несколько возможных (из-за отсутствия учета приоритета операций) деревьев разбора для приведенной выше грамматики для арифметических выражений, для следующих цепочек. Подумайте, как можно устранить неоднозначность.

|  |  |
| --- | --- |
| Вариант | Цепочка |
| 1 | «10+2-4/5» |
| 2 | «2/(1+2)-4» |
| 3 | «2\*(1+2\*4)» |
| 4 | «2\*123+2» |
| 5 | «0\*12+20» |
| 6 | «(0-5\*9)+3» |

1. Задайте грамматику для чисел. Примеры чисел: 0, -5, 0.1, -2.24, -2.123e-15, -31.123E+7. Запись числа -31.123E+7 называется научной нотацией и её следует понимать так: .
2. Постройте грамматику определяющее то же множество, что и регулярное выражение:

|  |  |
| --- | --- |
| Вариант | Регулярное выражение |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |

1. Постройте деревья синтаксического разбора для приведенной программы (синтаксис и семантика приведены выше). Можно считать числа и имена переменных терминалами (не задавать для них грамматику). Какие значения будут принимать переменные после выполнения программы.

|  |  |
| --- | --- |
| Вариант | Программа |
| 1 | i=1; s=0;  while(i<8) {  s=s+i;  i=i+1;  } |
| 2 | i=1; s=0;  do {  s=s+i;  i=i+1;  } while(i<8) |
| 3 | i=1; p=1;  while(i<8) {  p=p\*i;  i=i+1;  } |
| 4 | a=1; b=1; i=0;  while(i<8) {  c = a + b; a = b; b = c;  i=i+1;  } |
| 5 | i=-3; s=0;  while (i<4) {  if (i<0) {s=s-i} else {s=s+i}  i=i+1;  } |
| 6 | i=-3; s=0;  do {  if (i<0) {s=s-i} else {s=s+i}  i=i+1;  } while (i<4) |

1. Выполните трансляцию приведенной в предыдущем пункте варианта программы в код стековой машины из параграфа 2.9.2. Дополните машину, если нужно, дополнительными командами.

## Кодирование знаний

Да, следующим шагом должно было быть описание кодирования знаний. Можно было рассказать о семантических сетях, логике предикатов, деревьях решений. И, конечно, привести описание форматов RDF (используемого в семантической паутине интернет: semantic web), формата PMML (Predicted Model Markup Language), предназначенного для обмена между системами извлечения знаний (Data Mining) и т.д. Но для этого нужна отдельная книга. Поэтому эти вопросы мы отдаем пытливому уму читателя на самостоятельное изучение.

1. Напомним русский язык. Первое лицо: я, мы, Второе лицо: ты, вы. И третье: он, она, оно, они. [↑](#footnote-ref-1)
2. **ГСН** метод – метод **Г**рубой **С**илы и **Н**евежества [↑](#footnote-ref-2)
3. Вероятность математическая – числовая характеристика степени возможности наступления какого-либо события в тех или иных определенных, могущих повторяться неограниченное число раз условиях. Характеризуется числом, заключенным между нулем и единицей [↑](#footnote-ref-3)
4. Звонок в службу поддержки оператора сотовой связи. Клиент: «Девушка, до меня не доходят sms сообщения!!!». Девушка: «Прочтите еще раз…». Бородатый анекдот. [↑](#footnote-ref-4)
5. См. цикл лингвистических сказок «Пуськи бятые» Петрушевской Людмилы Стефановны [↑](#footnote-ref-5)
6. © Раков Владимир Ильич, 2010 г. Возраст 2 года. [↑](#footnote-ref-6)
7. Мы, как и прежде, рассматриваем упрощенный случай *стационарного* источника, события которого совершенно не зависят от других событий и вероятности этих событий не изменяются в процессе работы источника. [↑](#footnote-ref-7)
8. Так, известно, что египетская иероглифическая нумерация возникла за 2500-3000 лет до н. э. [↑](#footnote-ref-8)
9. Замечательной особенностью позиционных систем счисления является то, что, используя конечный алфавит цифр, можно закодировать *сколь угодно большое число*. Вспомните, например, римскую систему счисления (она не является позиционной!) с цифрами (I→1, V→5, X→10, L→50, C→100, D→500, M→1000). Для представления числа 4000 уже приходится изобретать новый символ (римляне вводили подчеркивание, что было эквивалентно увеличению числа, соответствующего цифре в 1000 раз): . Для возникшей необходимости представления очень-очень большого числа (с которым столкнулись впервые) придется придумать очень-очень много символов и договориться со всем миром об их соответствии числам, что очень-очень неудобно. [↑](#footnote-ref-9)
10. Есть и формат с плавающей точкой, в котором сохраняется информация о том, между какими разрядами находится точка [↑](#footnote-ref-10)
11. Инвертируются все разряды числа , кроме самого младшего единичного [↑](#footnote-ref-11)
12. По крайней мере, по сравнению с дополнительным кодом. В стародавние времена эта простота непосредственно выражалась в экономии аппаратных затрат на создание вычислительных устройств и энергии потребления в процессе их эксплуатации [↑](#footnote-ref-12)
13. ***Int***egrated ***el***ectronics. Название фирмы Intel – аббревиатура от «Интегральная электроника» [↑](#footnote-ref-13)
14. Да, по сети данные передаются именно в таком (big-endian) формате. Действительно важно, чтобы 32 разрядное число от хоста на Intel пришло таким же по значению на хост на Motorola. Поэтому при передаче (получении) данных по сети хосты little-endian вынуждены делать простые преобразования порядка следования байт (иногда это делает сетевой адаптер совершенно прозрачно для процессора). [↑](#footnote-ref-14)
15. Intel Architecture. 64-разрядная линейка процессоров [↑](#footnote-ref-15)
16. Совершенно справедлив вопрос: «А как же ноль?». См. далее [↑](#footnote-ref-16)
17. Согласно легенде о вавилонской башне, Бог таким образом лишил взаимопонимания членов одного из самых великих строительных коллективов. [↑](#footnote-ref-17)
18. Пока официально не принят ни один такой язык. Общественность тяготеет к естественным языкам, мирясь с их сложностью в пользу большей одухотворенности. [↑](#footnote-ref-18)
19. Впрочем, наши древние славянские «черты и резы» на бересте вполне подходили для восприятия с помощью осязания. [↑](#footnote-ref-19)
20. Да, это не новомодное понятие, обозначающее графические обозначения объектов операционной системы в графическом интерфейсе пользователя. [↑](#footnote-ref-20)
21. Это, впрочем, не совсем так. Буквой текста может кодироваться, например, слог (набор звуков) и т.д. [↑](#footnote-ref-21)
22. За единицу времени принимается длительность одной точки. Тире – три единицы. Промежуток между точками или тире в одном символе – одна единица. Промежуток между символами одного слова – три единицы. Промежуток между словами – семь единиц. [↑](#footnote-ref-22)
23. Далее мы увидим, что возможно, но очень сложно и дорого [↑](#footnote-ref-23)
24. Да и, если задуматься, возникают сомнения – существует ли он вообще? [↑](#footnote-ref-24)
25. Мифологическое существо с двумя лицами. Становясь между двух собеседников, корректировал черты обеих лиц и полностью контролировал дальнейшее общение. [↑](#footnote-ref-25)
26. Результат вычисления остатка есть целое число , для которого справедливо: ; – целые числа. [↑](#footnote-ref-26)
27. При современных вычислительных возможностях по крайней мере не менее [↑](#footnote-ref-27)
28. Простым называется целое число, неразложимое на целые сомножители [↑](#footnote-ref-28)
29. Образующим элементом называется элемент , степенью которого можно представить любое целое число из Кому интересно как это сделать обратитесь за теорией к хорошему учебнику по алгебре [↑](#footnote-ref-29)
30. В популярной телевизионной передаче «Кабачок «13 стульев»» показали сценку: пан Гималайский (артист Рудин) привез верблюдов и сопроводительное письмо. В письме было написано: "Направляем Вам двух одногорбых верблюдов и Гималайского". Вот у пана Гималайского и допытывались, где же третий верблюд, гималайский. Там и прозвучало: "Докажите, что Вы не верблюд" [↑](#footnote-ref-30)
31. Практика показывает, что это невозможно! [↑](#footnote-ref-31)
32. На самом деле стандарт на HTML (язык разметки гипертекста) был определен раньше, но это лишь специализация XML для отображения гипертекста в web-браузере. [↑](#footnote-ref-32)
33. В Unix/Linux LaTeX обычно установлен, а в Windows можно установить популярные бесплатные MikTeX или TeX Live. Для Mac можно использовать MacTeX. Есть масса других вариантов. Это *не реклама*. [↑](#footnote-ref-33)
34. См. например книгу: Д.Э. Кнут. Компьютерная типография [↑](#footnote-ref-34)
35. What you see is what you get – то что видишь на экране, получишь и на бумаге [↑](#footnote-ref-35)
36. Portable document format – переносимый формат печатных документов от компании Adobe. Подавляющее большинство технических документов распространяются в этом формате. [↑](#footnote-ref-36)
37. Comprehensive TeX Archive Network – всеобъемлющая сеть архивов TeX (Latex). Шаблоны, исходные тексты и программы для пользователей TeX/LaTeX на любой вкус. [↑](#footnote-ref-37)
38. Современная энциклопедия Wikipedia, в создании статей которой может участвовать любой желающий, при правке статей формулы вводятся в формате LaTeX. [↑](#footnote-ref-38)
39. ©: Изображен студент ВятГУ Максим Козлов. Съемка – студенты ВятГУ Кошкина Инна и Шишкина Елена. 2010 год. [↑](#footnote-ref-39)
40. Нет такого технического устройства, которое бы могло воплотить CIE-XYZ [↑](#footnote-ref-40)
41. Формат для обмена данными между музыкальными инструментами. Кодируются, например, нажатия клавиш, настройки громкости, выбор тембра, тональности, темпа и т.д. [↑](#footnote-ref-41)