О.В. Знаменская
С.В. Знаменский
Д.Е. Лейнартас
В.М. Трутнев

МАТЕМАТИЧЕСКАЯ ТИПОГРАФИЯ

Курс лекций

Красноярск

2008

Оглавление

Раздел 1. LaTeX — технология подготовки научного текста для публикации **6**

Лекция 1. Тех и другие форматы представления научной публикации 6

Настоящее и будущее TeX'нологии (6). Достоинства TeX'нологии (8). Как создавалась издательская система LaTeX 2ε (10). TeX'нические реализации (12). Программистам на заметку (13). Дистрибутивы LaTeX (15)

Лекция 2. Классы и пакеты документов в LaTeX 17

Структура исходного файла в LaTeX (17). Пакеты и классы документов в LaTeX (19)

Лекция 3. Поддержка письменностей и национальных типографских особенностей 30

Кодировка шрифта в LaTeX (31). Поддержка переносов в LaTeX. Двуязычные документы (36). Особенности русскоязычного набора (38)

Лекция 4. Управление шрифтовыми выделениями и подключение шрифтов 44

Классификация наборных шрифтов (Typeface classification) (44). Выбор шрифтов и их атрибутов в LaTeX (48). Коллекции кириллических шрифтов (Cyrillic fonts) в LaTeX (55). Как выбрать подходящий шрифт? (68)

Лекция 5. Расширенные возможности набора математического текста 70

Тонкости набора формул (71). Матрицы (89). Коммутативные диаграммы (97). Окружения типа теоремы (100)

Лекция 6. Библиография и цитирование литературы 105

Оформление ссылок на литературу в исходном файле (105). Создание списка литературы с использованием библиографической базы данных BiBTeX (108). Полезные советы (111)

Лекция 7. Управление заголовками. Создание указателей. Стиль автоматически формируемого содержания 112

Управление заголовками (112). Стиль автоматически формируемого содержания (114)

Раздел 2. Основы программирования в TEX и LATEX **119**

Лекция 8. TeX как язык программирования 119

Категории символов (119). Коды символов (122). Правила чтения исходного файла (123)

Лекция 9. Счетчики, длины, боксы 126

Размеры (126). Счетчики (128). Боксы (130). Клей (131)

Лекция 10. Таблицы в LATEX 133

Строение таблицы (133). Возможности окружения tabular (135). Пакеты для оформления элементов таблицы (138). Таблицы, занимающие несколько страниц (147). Импорт содержимого таблицы из базы данных (150)

Лекция 11. Управление формой абзацев, бегущие заголовки и плавающие вставки 152

Управление формой абзацев (152). Размещение плавающих объектов (162)

Лекция 12. Управление компоновкой страницы. Стиль страницы . . 166

Полоса набора (166). Настройка макета (169). Печать в две колонки (172)

Лекция 13. Создание собственных стилевых файлов 174

Создание классов и пакетов (175). Структура класса и пакета (179). Команды для создания классов и пакетов (186)

Лекция 14. Принципы трансляции исходных файлов. Язык PostScript 200

Принципы трансляции исходных файлов (200). Язык PostScript (203). Поддержка языка PostScript в TEX. Пакет PSTricks (208)

Лекция 15. Создание новых команд. Программирование на языке TEX 215

Определения и переопределения в LATEX (215). Макросы TEX (217)

Раздел 3. Программирование презентационных эффектов 223

Лекция 16. Управление цветом, графический драйвер 223

Пакет color (223). Пакет colortbl (227)

Лекция 17. Подготовка и вставка плавающих объектов, обтекание текстом 231

Пакет floatflt (231). Пакет wrapfig (232). Класс NCC (234)

Лекция 18. Создание документов в формате PDF. Проект PDFTEX . . 237

Информация о документе (237). Включение графики в PDF-документы (239). Гиперссылки. Пакет hyperref (240)

Лекция 19. Подготовка качественных иллюстраций для LATEX и их вставка 243

Пакеты graphics и grapicx (243). Ряд команд пакета graphics (248)

Лекция 20. Гипертекст в PDF 250

Команды, определенные в пакете hyperref (250). Опции пакета (251)

Лекция 21. Создание презентаций в LATEX'е 259

Пакет Beamer (261). Пакет PowerDot (277). Пакет Prosper (282). Несколько полезных советов (284)

Лекция 22. Электронные тесты и генерация вариантов заданий . . . 285

Создание электронных тестовых заданий средствами acrotex (285). Возможности пакета fp для генерации учебных заданий (292)

Раздел 4. Графика в LaTeX **302**

Лекция 23. METAPOST и TeX 302

METAPOST и TeX (302). Кривые (306)

Лекция 24. Пакет MFPic 314

Окружение mfpic (315). Гистограмма и секторная диаграмма (323). Префиксные макрокоманды (331). Аффинное (линейное) преобразование координатной системы (341). Функции (343). Метки и заголовки (347). Параметры (350). Цветная графика в MFPIC (353)

Лекция 25. Язык METAPOST'а 355

Типы данных (355). Операторы (357). Переменные (358). Линейные уравнения (359). Уравнения и координатные пары (360). Примеры (362)

Лекция 26. Кривые в METAPOST'е 370

Основные команды и приемы (370). Примеры (375)

Лекция 27. Стили рисования и модификация рисунков 378

Пунктирные линии (378). Аффинные преобразования (379). Стрелки (383). Стандартные кривые (385). Перья (386). Построение циклов (387)

Лекция 28. Программирование. Циклы и условные операторы. Макросы 391

Макросы (391). Параметризованные макросы (394). Макросы vardef (395). Циклы (396)

Лекция 29. Взаимодействие METAPOST'а и LaTeX'а 401

Интеграция текста и графики (401). Взаимодействие METAPOST'а и LaTeX'а (404)

Лекция 30. 3D графика и мультипликация 409

Поддержка трехмерной графики в LaTeX (409). LaTeX и мультипликация (415)

Литература **421**

Раздел 1. LaTeX — технология подготовки научного текста для публикации

Лекция 1

Тех и другие форматы представления научной публикации

Краткая история TeX'а и LaTeX'а. LaTeX и его составляющие. CTAN — Comprehensive TeX Archive Network (всеобъемлющая сеть архивов TeX). Основные дистрибутивы, программы. Основные форматы представления научной публикации и возможности преобразования один в другой.

1. Настоящее и будущее TeX'нологии

1.1. Настоящее

Широко известен факт, что для верстки текстов с большим количеством математических и химических формул удобно использовать издательскую систему LaTeX, которая работает на базе специализированного языка программирования TeX. Более того, в настоящее время формат TeX признан мировым стандартом подготовки научных публикаций, в которых интенсивно используются формулы, схемы, графики, рисунки и диаграммы. Система позволяет готовить публикации на английском, русском, французском, немецком, итальянском, испанском и многих других языках с возможностью перехода с одного языка на другой в любом фрагменте текста.

Издательская система LaTeX является образцом, к которому должна стремиться любая другая издательская система. Этот тезис в том или ином виде можно встретить во многих изданиях, посвященных описанию пакета LaTeX. В подтверждение можно привести официальное мнение пользовате-

лей TEX из США: “*Вопрос использования TEX может послужить основанием для религиозных войн, т. к. те, кто используют его возможности для подготовки математических книг и статей, считают, что никакое сочетание других имеющихся средств не сможет обеспечить производство высококачественных полиграфических изданий при подготовке их самим автором*” (цитата из документа “Just What Is TEX?” главного сайта TUG—TEX Users Group).

Относительно медленное распространение TEX как средства для научных публикаций в России означает лишь плохое знакомство с возможностями этой системы, во многом идеальной для своей области применения. (А ведь TEX произносится не как «текс», а именно как «тех», т. е. по-русски!)

В настоящее время почти все научные журналы по математике, логике, физике как за рубежом, так и в России, принимают к рассмотрению статьи, набранные с помощью системы LATEX, более того, LATEX принят как стандарт для набора печатных изданий многими ведущими издательствами (в том числе, издательствами «Мир» и Addison-Wasley).

1.2. Взгляд в будущее

Уже достаточно распространена практика конвертирования всех текстов, включая презентации, создаваемые в LATEX, в формат `PDF`.

Portable Document Format (`PDF`) — один из форматов представления документов в электронном виде. Главной его особенностью является кроссплатформенность. Это означает, что документ `PDF` будет выглядеть одинаково при его просмотре на любом `ПО` в любой операционной системе или при распечатке на любом оборудовании, что вполне соответствует идеологии TEX. Файл в формате `PDF` намного меньше размером, чем `Postscript`, позволяет осуществлять поиск по тексту, позволяет включать изображения в формате `PNG` или `JPG`, а также делать гиперссылки, встроенное оглавление и многое другое.

Немаловажно, что формат `PDF` получил сегодня огромное распространение. Де-факто он стал самым настоящим стандартом для обмена элек-

тронными документами. Многим людям приходится сталкиваться с различными задачами, касающимися формата PDF: созданием и чтением файлов, заполнением форм, редактированием и комментированием документов. Поэтому не случайно последние несколько лет пользователей LaTeX 2_ε, активно создающих электронные публикации в выходном формате PDF, интересуют следующие вопросы:

1. Можно ли в LaTeX вставлять скрипты, написанные, например, на языке JavaScript?

2. Можно ли с помощью LaTeX вставлять в pdf-файлы видео?

3. Можно запрограммировать с помощью LaTeX в pdf-файле всплывающее окно с каким-либо содержанием (тем же видео-роликом, или линком на сайт)?

TeX и LaTeX 2_ε в классическом виде этих возможностей не поддерживают. Однако в pdfTeX'е, предназначенном для генерации pdf-документов, в том числе интерактивных, совсем недавно (год или два назад) такие возможности стали появляться, и некоторые из них станут предметом нашего рассмотрения в настоящих лекциях.

2. Достоинства TeX'нологии

Мы остановимся на основных преимуществах LaTeX 2_εперед большинством современных текстовых процессоров типа WYSIWYG (What you see is what you get), таких как MS Word или Corel WordPerfect, позволяющих в процессе работы видеть на экране выводимый на печать результат.

Заметим, что при использовании LaTeX обычно невозможно увидеть итоговую картину во время печатания текста. Ее, однако, можно посмотреть на экране после обработки файла LaTeX.

- Любой пользователь может легко изготовить профессионально выполненные макеты, делающие документы действительно выглядящими «как изданные».

- Если пользователь не ставит задачи разработки собственного дизайна книги, то ему достаточно знать лишь несколько понятных команд, задающих логическую структуру документа. Ему практически никогда не нужно возиться собственно с макетом документа. И уж тем более не нужно знать сам язык программирования TEX.

- LATEX поощряет авторов писать хорошо структурированные документы, так как именно так LATEX и работает — путем спецификации структуры. Неструктурированные и неорганизованные документы в этой системе писать сложно, и это можно расценивать как одно из ее достоинств.

- Удобно поддержана верстка математических формул.

- Легко изготавливаются даже сложные структуры, типа примечаний, оглавлений, библиографий и прочее.

- Для решения многих типографских задач, не поддерживаемых напрямую базовым LATEX, есть свободно распространяемые дополнительные пакеты. Например, существуют пакеты для включения PostScript графики или для верстки библиографий в точном соответствии с конкретными стандартами.

- TEX, форматирующее сердце LATEX 2_ε, чрезвычайно мобилен и свободно доступен. Поэтому система работает практически на всех существующих платформах.

Итак, TEX-система дает полный контроль над структурой будущего документа, однако не создает иллюзий относительно того, что сложные проблемы могут быть решены просто: хотя предопределенные макеты имеют множество настраиваемых параметров, создание полностью нового макета документа «с нуля» может оказаться непростой задачей и займет немало времени.

3. Как создавалась издательская система LaTeX 2ε

3.1. Кнут и TeX

Создателем ядра обсуждаемой издательской системы является американский программист и математик из Стэнфордского университета Дональд Кнут (Donald E. Knuth). Он хорошо известен как автор пятитомного труда «Искусство программирования» (The Art of Computer Programming).

TeX — это компьютерная программа, созданная Дональдом Кнутом. Она предназначена для верстки текста и математических формул.

Легенда гласит, что Кнут написал программу TeXдля того, чтобы подготовить к печати свои книги серии «Искусство программирования для ЭВМ», поскольку был крайне недоволен тем,что́ Американское Математическое Общество делало с его работами в процессе их публикации. Первая версия TeX'а появилась в 1978 году.

TeX, в том виде, в каком мы его сегодня используем, был выпущен в 1982 г. Начиная с 1989 года, во избежание распространения несовместимых версий, все изменения в TeX'е сводятся в основном к исправлению ошибок. Последние несколько лет TeX стал чрезвычайно стабилен и практически не содержит ошибок. Номер версии TeX сходится к π и сейчас равен 3.14159.

3.2. Появление макропакетов

Если получить программу TeX из кнутовского исходного текста, то есть перевести псевдокод в `Pascal` или C, а затем скомпилировать, то использовать полученную программу для обработки текстов будет еще невозможно, поскольку «чистый TeX» не знает смысла ни спецзнаков, ни макросов. В него встроены только «примитивные команды». Набор TeX'овских макросов, используемый для обработки текстов, называется макропакетом.

Макропакет Plain TeX

Первый макропакет написал сам Кнут одновременно с программой TeX, он называется Plain TeX, и подробно описан в книге “The TeXbook”.

Эта книга является справочником по TEX‘у, соответствующий файл `texbook.tex` в открытом доступе содержится в любом современном дистрибутиве TEX’а.

Макропакет $\mathcal{A}\mathcal{M}\mathcal{S}$TEX

Следующим после Plain TEX’а макропакетом, получившим распространение, стал $\mathcal{A}\mathcal{M}\mathcal{S}$TEX, созданный Майклом Спиваком (Michael Spivak). По сравнению с Plain TEX’ом в него были добавлены удобные средства для набора сложных формул, особенно многострочных. Подчеркнем, что всех $\mathcal{A}\mathcal{M}\mathcal{S}$TEX’овских эффектов при достаточном упорстве можно добиться и в Plain TEX’е, но Спивак создал для этих целей макроопределения, которыми математики с удовольствием пользуются.

В настоящее время $\mathcal{A}\mathcal{M}\mathcal{S}$TEXможно рассматривать как англоязычную издательскую систему, ориентированную на узкий круг приложений — верстку научных статей для математических журналов и книг.

Макропакет LATEX–LATEX 2_ε

LATEX — макропакет, позволяющий авторам верстать и печатать их работы с высоким типографским качеством, при помощи заранее определенных, профессиональных макетов. В качестве механизма для верстки он использует TEX. Возможно, одной из важнейших новых черт LATEX’а явилась возможность автоматической нумерации и — главное — автоматической генерации ссылок с помощью команд `\label`, `\ref` и `\pageref`. Достижение этого эффекта средствами Plain TEX’а требует весьма изощренного программирования.

Макропакет LATEX был написан Лэсли Лампортом (Leslie Lamport) в 1984 году, а вторая версия LATEX 2.09 вышла в 1989 году.

В 1994–1995 годах пакет LATEX был обновлен. В нем были сделаны некоторые давно ожидавшиеся улучшения, объединены все варианты LATEX, разошедшиеся со времени выпуска версии LATEX 2.09. Новая версия получила название LATEX 2_ε.

Среди прочего была решена задача включить в LATEX возможности $\mathcal{A}\mathcal{M}\mathcal{S}$TEX’а, в частности, возможность пользоваться символьными шриф-

тами Американского математического общества.

В настоящее время поддержкой и совершенствованием системы LaTeX занимается группа TeX'нологов во главе с Frank Mittelbach. В настоящее время LaTeX практически стабилизировался, хотя раз в год и выходит новая версия.

Макропакет ConTeXt

Весьма интересным, появившимся в конце 1990-х, макропакетом является ConTeXt. Он позволяет, в частности, легко создавать цветные интерактивные `pdf`-документы и печатать сложные химические структуры. Пока ConTeXt не поддерживает в полном объеме работу с текстами на русском языке.

4. TeX'нические реализации

Программа TeX (то есть интерпретатор языка TeX) написана Кнутом не на каком-то из использовавшихся в то время языков программирования, но на специальном псевдокоде.

Благодаря этому TeX можно сравнительно легко приспособить к различным типам компьютеров и операционных систем, и эти реализации будут работать одинаково.

Конечно, пользовательский интерфейс и интерфейс с операционной системой в разных реализациях устроены по-разному, но при обработке одного и того же `tex`-файла с одним и тем же макропакетом на любой платформе получится один и тот же `dvi`-файл, при печати которого получится один и тот же текст.

4.1. А у тебя LaTeXили MiKTeX?

Различные реализации TeX'а имеют свои названия.

Долгое время была популярна созданная Эберхардом Маттесом (Eberhard Mattes)реализация TeX'а для `DOS`, называемая `emTeX`.

Для `Windows` существуют две реализации `fpTeX` и `MiKTeX`, последняя пользуется наибольшей популярностью.

Современная реализация TeX'а для `UNIX` (для `Linux`, `FreeBSD` и др.) называется `teTeX`.

Для Макинтошей существуют по крайней мере две реализации TEX'а: коммерческая, называемая `TeXtures`, и shareware-вариант `OzTeX`.

Не следует путать реализации с макропакетами. В любой из реализаций TEX'а можно установить любой макропакет, `Plain`, `LaTeX`, `AmSTeX` или тот, который вы когда-нибудь создадите сами.

Подведем итог: LATEX — это макропакет TEX, а MiKTEX— реализация TEX под `Windows`. Поэтому на вопрос новичков-TEX'нологов: «А у тебя LATEXили MiKTEX?» ответить невозможно.

5. Программистам на заметку

Ядро системы — процессор TEX является машинно-независимым языком форматирования полиграфических документов. Автор TEX'а определяет его как «*инструмент для превращения набранного компьютерным образом манускрипта в документ, полиграфическое качество которого сопоставимо с тем, что дают самые современные печатающие устройства*».

Программа TEX распространяется свободно, а ее исходные тексты полностью доступны. Текст программы TEX написан на изобретенном Кнутом псевдокоде под названием `Web`; раньше `Web` транслировали в `Pascal` (с помощью программы, написанной все тем же Кнутом), в настоящее время используется транслятор из `Web`'а в язык C.

Помимо TEX, фундаментальными компонентами любой TEX-системы являются программы: `mf` (METAFONT), `bibtex` (BibTEX), `mpost` (METAPOST) — их исходники на языке `Web` (транслируется в Паскаль и затем, если нужно, в C++ перед компиляцией), доступны свободно в `CTAN:/systems/knuth`. Используя `Web`, можно получить одновременно программу и отличную документацию к ней.

TEX — это компилирующийся универсальный язык программирования. Алгоритм на TEX записывается, используя средства, подобные препроцессору C++, т. е. макроподстановки, условную трансляцию, включение файлов и т. п., к которым добавлены возможности для работы с переменными и файлами. Файловые операции включают в себя только последовательные

чтение и запись. Синтаксически TEX-текст состоит из собственно текста, макросов и примитивных команд, причем последние внешне не различаются и выглядят за исключением нескольких особых случаев как последовательности, начинающиеся со специального `escape`-символа, за которым следуют либо буквы, либо одиночный символ.

В частности, в Plain TEXи LATEX`escape`-символом является знак обратной наклонной черты (`backslash`).

Макросы не обязательно вызывать явно — они могут вызываться сами в определенных контекстах, подобно конструкторам/деструкторам C++.

Недостатком TEX является отсутствие средств для вызова других программ. Например, при подготовке указателей к книге список строк нужно отсортировывать. Выполнять сортировку средствами самого TEX слишком громоздко и неэффективно. Гораздо лучшим решением, было бы вызвать внешнее средство сортировки, например, программу `sort`, но это невозможно сделать изнутри TEX-программы. На практике подобные задачи (среди которых стоит выделить работу с библиографией при помощи BibTEX и построение указателей при помощи makeindex) решаются двумя прогонами TEX'а, между которыми к данным применяются внешние средства.

Внесение в простой текст (plain text) дополнительной информации об его оформлении или структуре осуществляется при помощи разметки текста (markup). TEX после подключения макроопределений задает разметку текста входных документов. Различают физическую (или процедурную) и логическую разметки.

При физической разметке точно указывается, что нужно сделать с выбранным фрагментом текста: показать курсивным, приподнять, центрировать, сжать, подчеркнуть и т. п. При логической разметке указывается структурный смысл выбранного фрагмента: примечание, начало раздела, конец подраздела, ссылка на другой фрагмент и т. п.

Логическую разметку всегда можно преобразовать в физическую, используя выбранный стиль. Имея наборы документов в логической разметке можно всегда при печати придавать им наиболее привлекательный вид,

своевременно получая от специалистов-дизайнеров новейшие стили. Преобразование физической разметки в логическую формальными средствами практически невозможно.

Макропакеты Plain и Eplain TEX задают процедурную разметку, а LATEX, ConTEXt и TEXinfo — логическую. Последнее разделение довольно условно, поскольку в Plain TEX можно определить формат, соответствующий хорошей логической разметке, а используя LATEX можно применять в любых количествах средства физической разметки.

METAFONT также создан Кнутом и используется для создания высококачественных растровых (bitmap) шрифтов. Шрифты на этом языке масштабируемые, растр знаков создается для конкретных устройств вывода, но не в реальном времени интерпретацией как для шрифтов Adobe PostScript или TrueType, а предварительной компиляцией, т. е. METAFONT — это еще и растеризатор (raster image processor, RIP).

Кнут «заморозил» развитие METAFONT и поэтому эта программная система, имеющая возможности для развития и модификации не меньшие, чем сам TEX, практически не имеет вариантов.

Использование фирменных шрифтов с TEX осложнено тем, что в них часто не определен полный диапазон нужных знаков. Эту проблему снимают виртуальные шрифты, позволяющие объединять в себе знаки из разных шрифтовых файлов и задавать нужную кодировку. Первые виртуальные шрифты для TEX (Computern Modern) нарисовал сам Кнут.

Программа `Dvips` обеспечивает подключение к TEX возможностей языка `PostScript`, которые значительно превосходят возможности языка `dvi`-файлов Д. Кнута, а программа `Ghostscript` позволяет как печатать `ps`-файлы? так и конвертировать их в другие форматы, например, в `pdf` или растровые картинки. Работа с масштабируемыми не `mf`-шрифтами основана на `Dvips` и `Ghostscript`.

6. Дистрибутивы LATEX

LATEX не является монолитной программой, а состоит из набора пакетов/программ, причём набор пакетов не фиксирован, что позволяет создавать

дистрибутивы, преследующие ту или иную цель.

На сегодня все дистрибутивы LaTeX имеют общий корень, который носит название CTAN или The Comprehensive TeX Archive Network (http://www.ctan.org). CTAN — это репозитарий, в который стекаются все разумные наработки в области совершенствования TeX. Он представляет из себя множество зеркалируемых серверов по всему миру. Например, наиболее известным в среде GNU/Linux является дистрибутив teTeX2. Этот дистрибутив был создан Томасом Эшером (Thomas Esser). Первая публичная версия в 1994 умещалась на три дискетки. В мае этого (2006) года Томас принял решение о прекращении поддержки своего детища в пользу дистрибутива TEX Live. Мантейнеры дистрибутива используют определение понятия свободы, принятое Фондом открытого программного обеспечения (http://www.gnu.org/philosophy/free-sw.html — Free Software Foundation's definition) или сообществом Debian (http://www.debian.org/social_contract — Debian Free Software Guidelines).

Лекция 2 ⏮

Классы и пакеты документов в LaTeX

Исходный файл в LaTeX, подключение классов и пакетов в преамбуле файла. Базовые и специализированные классы и пакеты.

1. Структура исходного файла в LaTeX

Для компиляции LaTeXготовится исходный файл, который можно набрать в любом текстовом редакторе (хотя специальные редакторы значительно облегчают набор) и сохранить с расширением `.tex` (не `.txt` !).

Структура исходного файла проста. Она состоит из тела документа и преамбулы, в которой указывается, какие средства должен задействовать LaTeX, чтобы корректно обработать документ. Таким образом, преамбула отвечает за стиль оформления, а тело — за содержание документа. Минимальный документ LaTeX на русском языке можно получить, скомпилировав обычный текстовый файл с расширением `.tex` и со следующими строчками:

```
\documentclass{article}
\usepackage[russian]{babel}
\begin{document}
Крошечный документ.
 \end{document}
```

После компиляции получится следующий текст:

Крошечный документ.

Класс документа — это первое, что требуется указать при наборе. Класс документа выбирается с помощью команды

```
\documentclass[необязательные параметры]{имя класса}.
```

Ее необходимо указать ровно один раз в самом начале документа.

В качестве обязательного аргумента **имя класса** указывается имя класса или, что тоже самое, стиля (в нашем случае `article`), которому через запятую передаются необязательные параметры. В целях совместимости в стандартных (и других) классах используются одинаковые необязательные параметры. Вот некоторые наиболее полезные:

- `10pt`, либо 11pt, 12pt — установка базового размера шрифта.
- `a4paper` — установка размера листа бумаги. По умолчанию LaTeXиспользует размер листа `letter`.
- `draft` — режим черновой печати для «отлавливания» проблем вёрстки. В этом режиме не внедряются картинки (вместо них вставляются прямоугольники нужного размера) и отмечаются строчки, где алгоритм разбиения абзаца на строки даёт осечку.
- `oneside|twoside` — форматирование документа для односторонней и двухсторонней печати, соответственно.
- `twocolumn` — печать в две колонки.

Преамбула документа

В LaTeX'е часто требуется применять специфические настройки, отличные от настроек, принятых по умолчанию (поля, списки и таблицы, библиографические ссылки и прочее).

Для этого используются пакеты расширений, подключаемые в преамбуле (или в «шапке»), LaTeX'овского документа.

Приведем пример такой преамбулы с указанием на действия пакетов. Преамбула подходит для подготовки документов с сильным физико-математическим уклоном, например, для вёрстки статей и отчётов с обильным использованием формул, рисунков, таблиц и ссылок на литературу:

```
\documentclass[a4paper,12pt]{article}
%поддержка русского языка в Latex:
\usepackage[T2A]{fontenc}
\usepackage[cp1251]{inputenc}
```

```
\usepackage[russian]{babel}
%расширенные наборы математических символов и теорем:
\usepackage{amssymb,amsmath,amsfonts,latexsym,mathtext}
\usepackage{indentfirst}%делать отступ в начале парагр.
\usepackage{tabularx} %продвинутые таблицы
\renewcommand{\baselinestretch}{1.24} %интервал
%разрешить включение PostScript-графики
\usepackage[dvips]{graphicx}
%относительный путь к каталогу с рисунками
\graphicspath{{edgeimages/}}
```

2. Пакеты и классы документов в LaTeX

2.1. Базовые классы документов в LaTeX

Исторически LaTeXсоздавался для облегчения создания разного рода *математических* текстов. Какую печатную продукцию создает математик? В первую очередь это статьи, книги, отчеты, доклады, письма и слайды для устных выступлений. Каждый из этих документов имеет свое назначения, стиль, правила оформления. Соответственно и в LaTeX'е содержится 6 базовых классов, которые определяют вид и структуру документа, отвечающего американским полиграфическим требованиям к данному типу публикации:

- `article (статья)` В этом классе определены команды, отвечающие за разбиение текста на пункты, параграфы, разделы (команды секционирования). Одним из элементов оформления является предисловие (окружение abstract).

- `book (книга)` В классе `book` присутствует расширенный набор команд секционирования, в который добавлена команда `chapter` (глава). В этом классе также присутствует базовый набор команд для оформления титульного листа, предисловия и оглавления.

- `report (отчёт)` Класс, предназначенный для создания отчётов, а

также дипломных работ и диссертаций, является упрощённой версией класса `book`.

- `proc (доклад)` Это класс, предназначенный для создания тезисов докладов. Он является модификацией класса `article`, основное отличие состоит в обязательной двухколоночной печати.

- `letter (письмо)` Этот класс создан для написания деловых писем. Набор команд в этом классе существенно отличается от уже перечисленных, в частности, для писем нет нужды в командах секционирования.

- `slides (слайды)` Позволяет создать слайды с минимальным необходимым для научного доклада оформлением.

Класс определяется в файле с расширением `.cls`. В классе задаётся геометрия страницы и определяются команды секционирования (разбиения на главы, параграфы, оформление оглавления, аннотация в начале статьи и т.д.). Можно варьировать настройки класса с помощью подключаемых стилевых файлов с расширением `.sty`.

2.2. Стандартные специализированные пакеты

При помощи базовых классов LaTeX можно создать документ, вполне удовлетворяющий стандартным требованиям.

Однако, как правило, при помощи базовых классов не удается или довольно сложно реализовать авторский замысел дизайна печатного (электронного) продукта, поэтому на основе базовых классов или «с нуля» создаются новые классы. Неудивительно, что количество уникальных файлов с расширением `.cls` в последних дистрибутивах LaTeXуже приближается к двум сотням.

Многие издательства и журналы предоставляют свои собственные стилевые файлы, что позволяет быстро оформить публикацию, соответствующую стандартам издания. Каждый серьёзный научный журнал и крупная конференция имеет свой LaTeX-класс. Как правило, этот класс лежит где-то на официальном сайте, например, журналы, издаваемые издательством

МАИК «Наука/Интерпериодика», должны следовать правилам выложенным здесь: http://www.maik.ru/pub/tex/.

Охарактеризуем наиболее употребимые пакеты классов документов.

Классы документов Американского математического общества

Американское математическое общество (AMS) распространяет пакет из трех специализированных классов документа, предназначенных для набора математических текстов: amsart, amsbook и amsproc, которые используются для материалов, публикуемых в журналах $\mathcal{AMS}$и Сибирском математическом журнале. Все $\mathcal{AMS}$'овские классы документов автоматически подключают стилевые пакеты amsmath и amsthm для оформления теорем и печати математических символов. Основные особенности этих классов относятся к титульной информации и рубрикации документа.

Пакет классов «extsizes»

Пакет extsizes содержит классы extarticle, extbook, extletter, extproc, extreport, по назначению аналогичные классам article, book, letter, proc, report. Базовый набор пакетов не позволяет использовать базовые размеры шрифта в 9pt, 14pt и 17pt. Этот конкретный недостаток ликвидирован в пакетах классов extsizes, которые отличаются от стандартных только возможностью указать базовый размер шрифта, отличный от обычного 10-12pt.

Пакет NCC для создания русскоязычных документов

Ещё один вариант замены стандартных классов — это пакет NCC.

Для использования следует загрузить класс ncc и передать ему желаемый стиль оформления.

Класс NCC содержит четыре стиля оформления документов, заказываемые обычными опциями article (статья), preprint (препринт), book (монография) и report (отчет), который эквивалентен стилю book с опцией oneside.

В классе `preprint` специальной командой задается номер препринта. Титул оформляется на отдельной странице. В классе `book` команда `\bookeditor{текст}` позволяет вставить текст под названием книги (обычно это научные редакторы книги). Опциии `russian` в `\documentclass[Список опций]{ncc}` вполне достаточно при наборе русского текста в стандартной кириллической кодировке Windows (CP1251), при этом автоматически реализуется русский стиль оформления заголовков и соблюдение других отечественных типографских особенностей. Имеются и другие возможности, не заложенные в базовых классах.

Информацию об использовании этого класса можно почерпнуть в краткой инструкции к пакету (`nccLaTeX.pdf`). О пакете `NCC` подробно написано в книге его создателя А. И. Роженко: «Искусство верстки в LaTeX'е», 2005 (ISBN 5-901548-25-6).

Классы для создания конструкторской документации

Конструкторская документация должна отвечать строгим стандартам оформления. Первая попытка разработать пакет, облегчающий ее создание, была предпринята В. Фёдоровым. В результате был создан пакет `eskd`, который для нормальной работы требует обязательной установки кириллических шрифтов из коллекции pscyr.

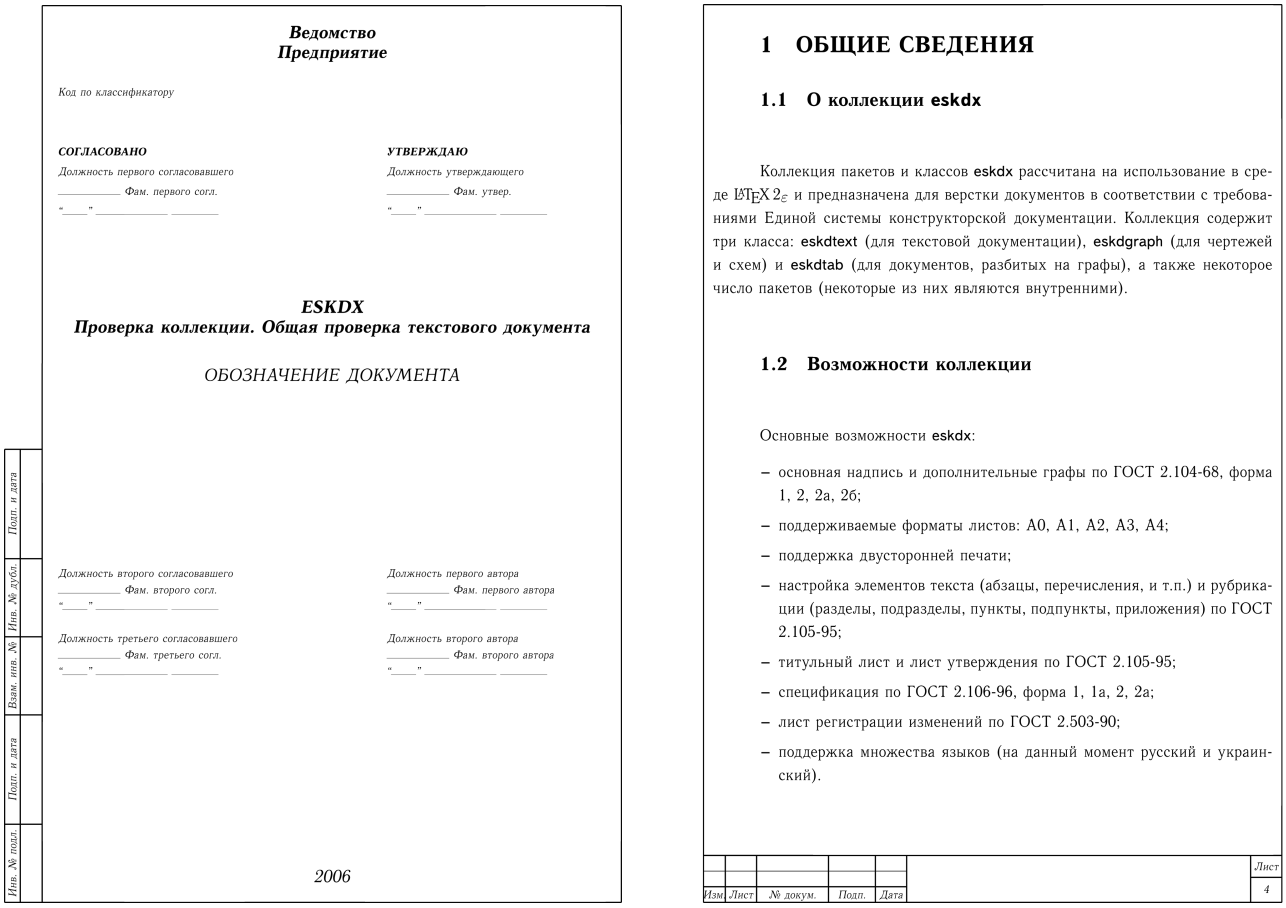
Более молодой пакет `eskdx` — это набор классов и стилей для LaTeX, предназначенный для верстки как текстовой, так и графической документации в соответствии с требованиями «Единой системы конструкторской документации». Его создал и и активно поддерживает К. Кориков.

Основу коллекции составляют три класса: `eskdtext` (для текстовой документации), `eskdbtab` (для чертежей и схем) и `eskdgraph` (для документов, разбитых на графы). Пакет снабжен понятной документацией на русском языке (`eskdx.pdf`).

К возможностям пакета относятся автоматическое создание основной надписи и дополнительных графов по ГОСТ 2.104-68, форма 1, 2, 2а, 2б; поддерживаемые форматы листов A0, A1, A2, A3, A4; поддержка двусторонней печати; поддержка русского и украинского языков (рис. 2.1).

Этот набор пакетов находится еще в стадии разработки, поэтому некоторые стандарты поддерживаются не полностью.

Пакет можно найти на CTAN в директории `CTAN/macros/LaTeX/contrib/eskdx/`.



Ведомство
Предприятие

Код по классификатору

СОГЛАСОВАНО
Должность первого согласовавшего
_____ Фам. первого согл.
“____” _____ _____

УТВЕРЖДАЮ
Должность утверждающего
_____ Фам. утвер.
“____” _____ _____

ESKDX
Проверка коллекции. Общая проверка текстового документа

ОБОЗНАЧЕНИЕ ДОКУМЕНТА

Должность второго согласовавшего
_____ Фам. второго согл.
“____” _____ _____

Должность первого автора
_____ Фам. первого автора
“____” _____ _____

Должность третьего согласовавшего
_____ Фам. третьего согл.
“____” _____ _____

Должность второго автора
_____ Фам. второго автора
“____” _____ _____

Подп. и дата | *Инв. № дубл.* | *Взам. инв. №* | *Подп. и дата* | *Инв. № подл.*

2006

1 ОБЩИЕ СВЕДЕНИЯ

1.1 О коллекции eskdx

Коллекция пакетов и классов eskdx рассчитана на использование в среде LaTeX 2_ε и предназначена для верстки документов в соответствии с требованиями Единой системы конструкторской документации. Коллекция содержит три класса: eskdtext (для текстовой документации), eskdgraph (для чертежей и схем) и eskdtab (для документов, разбитых на графы), а также некоторое число пакетов (некоторые из них являются внутренними).

1.2 Возможности коллекции

Основные возможности eskdx:

- основная надпись и дополнительные графы по ГОСТ 2.104-68, форма 1, 2, 2а, 2б;
- поддерживаемые форматы листов: A0, A1, A2, A3, A4;
- поддержка двусторонней печати;
- настройка элементов текста (абзацы, перечисления, и т.п.) и рубрикации (разделы, подразделы, пункты, подпункты, приложения) по ГОСТ 2.105-95;
- титульный лист и лист утверждения по ГОСТ 2.105-95;
- спецификация по ГОСТ 2.106-96, форма 1, 1а, 2, 2а;
- лист регистрации изменений по ГОСТ 2.503-90;
- поддержка множества языков (на данный момент русский и украинский).

Изм | *Лист* | *№ докум.* | *Подп.* | *Дата* | *Лист* 4

Рис: 2.1. Класс `eskdx` — пример титульного листа и страница из файла `eskdx.pdf`

Исходники, распакованные файлы пакетов и классов, руководство в формате PDF на русском и украинском языках, FAQ на русском и примеры оформления документов доступны по ссылке `http://lostclus.linux.kiev.ua/`.

2.3. Новые классы под разные задачи

Количество пользователей LaTeX продолжает неуклонно расти. Изначально система LaTeX создавалась как система для верстки математических и естественно научных текстов и поэтому приспособлена к высококачественной передаче таблиц, формул, диаграмм. В настоящее время эта издательская система начала активно использоваться для верстки учеб-

ников, книг по юриспруденции, критических изданий классиков, поэзии, журналов с параллельным расположением текстов на разных языках и т.д. Соответственно, пополняется количество классов и стилевых файлов.

Рис: 2.2. Класс `papertex` — пример от автора класса Ignacio Liopis

Так, например, «инновационный», созданный в 2007 г. пакет `papertex`, позволяет применять LaTeX для вёрстки *газет*, то есть для того, для чего LaTeX в проекте вовсе не предназначался. Сам пакет можно найти на любом CTAN-архиве в директории `CTAN/macros/LaTeX/contrib/papertex/`.

Попытка типологизировать все известные на настоящий момент классы была предпринята Евгением Балдиным.

Остановимся на разновидностях классов и пакетов (наборов классов), наиболее перспективных в отношении их использования в учебно-научном процессе.

Заметим, что структура письма или служебной записки не очень сложна, поэтому класс `letter` и его заменители практически не используются.

Подготовка научных статей

Стандартного класса `article` вполне достаточно для подготовки научной публикации. Для верстки окончательного варианта статьи, как правило, используются собственные стилевые файлы журнала, в котором планируется публикация. Некоторые журнальные классы можно найти в стандартном дистрибутиве LaTeX. Класс `amsart` уже упоминался выше. Из русскоязычных журналов — класс `sibjnm` «Сибирского журнала вычислительной математики». Класс `elsart` обязателен для подготовки журнальных публикаций в издательстве Elsevier. Класс `nature` позволит подготовить pdf-файл для журнала Nature.

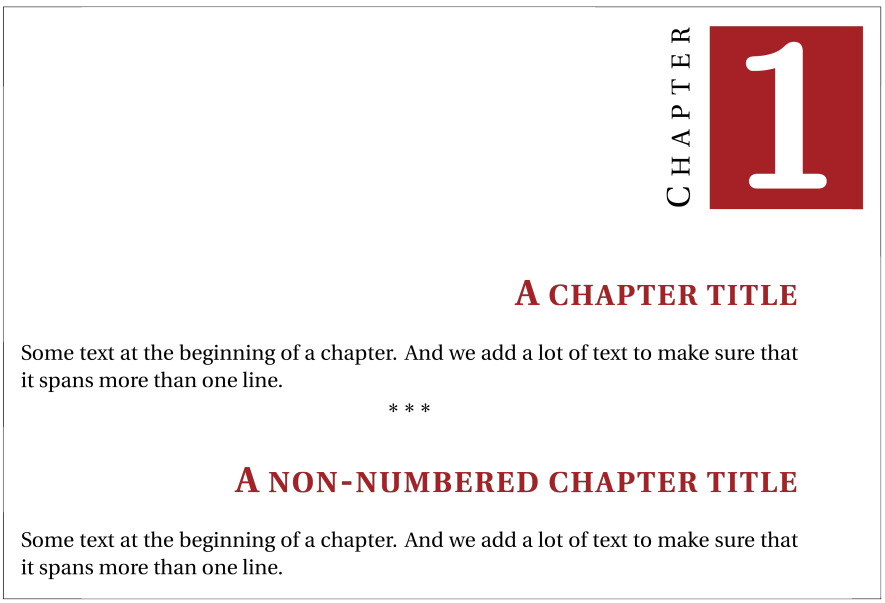
Верстка монографий, учебников и других книг

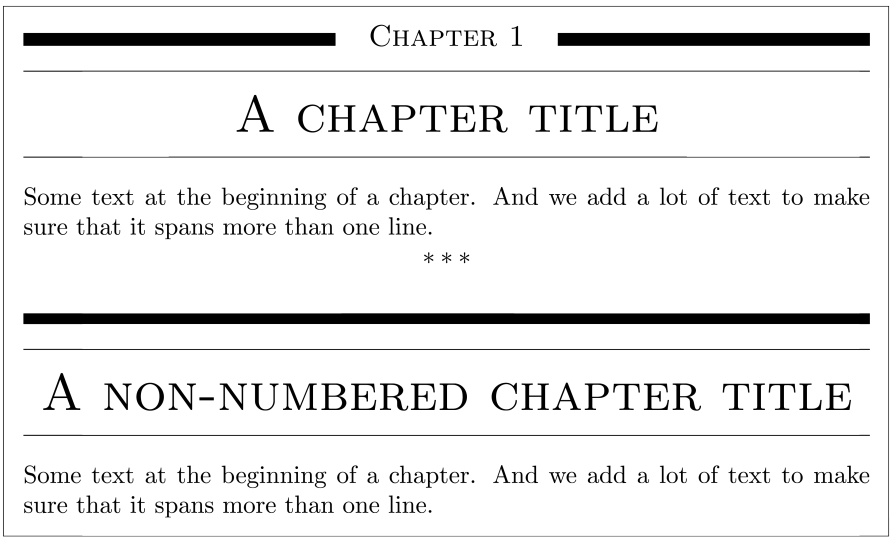
Написание книги — непростой процесс, и первоначальную «набивку» текста можно начать со стандартного класса `book`. Однако дизайн и структура книги может быть очень сложной, и тогда класс `book` как правило, оказывается неудобным. Он не позволяет, например, использовать базовые размеры шрифта в 9pt, 14pt и 17pt.

Этот конкретный недостаток, как уже отмечалось, ликвидирован в классе `extbook` из пакета классов `extsizes`.

Произведением TeX'нического искусства можно считать класс `memoir`, приспособленный для издания книг самого разного содержания. Как отмечает Е.Балдин, описание класса `memoir (memman.pdf)` представляет из себя книгу о создании книги, превышающую по объёму триста страниц (пока не переведенную на русский язык). Класс развивается до сих пор. Новые возможности описываются в дополнении к основной документации (`memmanadd.pdf`).

Варианты оформления глав книги в классе `memoir`:





Написанный специально для оформления данных лекций стилевой файл также существенно использует возможности класса `memoir`.

Подготовка отчетов

В отчете есть свои правила и структура. Стандарт оформления отчета существенно определяется организацией, для которой он создается. Большинство стандартных пакетов содержит аналоги класса `report`.

Подготовка курсовых, дипломных работ и диссертаций

Каждый университет имеет свой уникальный стиль оформления дипломных работ, существуют разные стили оформления диссертаций.

Можно пытаться использовать для этой задачи стандартный класс `book` (возникнет проблема набора основного текста шрифтом 14pt) или `extbook`, не свободный от остальных недостатков класса `book`.

Для диссертаций на английском языке можно использовать класс `ucthesis` от Калифорнийского университета (UC Berkeley). Для диссертаций на русском языке можно можно взять на `CTAN` в директории `CTAN/macros/LaTeX/contrib/disser` класс `disser`, созданный Станиславом Кручининым (без пользовательской документации).

Возможно после попыток использовать существующие классы Вам захочется написать свой.

Для создания настенного постера для защиты квалификационной работы можно воспользоваться пакетом `a0poster`, который позволяет работать с большими форматами бумаги или классом `sciposter` из одноимённого пакета.

Подготовка презентаций

С помощью стандартных классов `sides` и `seminar` может быть создана только очень простая презентация, состоящая из слайдов без всяких демонстрационных эффектов.

Класс `prosper` создан как улучшенный `seminar` и поддерживает не только оверлеи, гиперссылки и шаблоны оформления, но и «стандартный» набор динамических эффектов доступных через формат `pdf`. Вспомогательный класс `ppr-prv` позволяет создать печатную версию электронных слайдов класса `prosper`.

В пакет `texpower` входит класс `powersem`, который, по сути дела, просто загружает `slides`, а всю работу по созданию презентации выполняет стиль `texpower`. Возможности этого пакета сравнимы с `prosper` — просто он немного другой.

В последнее время наиболее популярен относительно молодой, но довольно мощный класс `beamer`, автор которого Тил Тантау (Till Tantau). Наличие широкого выбора стандартных тем позволяет быстро выбрать обрамление для презентации.

Класс `talk`, в отличие от упомянутых выше пакетов, позволяет пользователю определить более одного стиля слайдов для презентации. Резкая смена стиля во время доклада — иногда бывает нужно и такое.

Оформление сборников и трудов конференций

Для объединения разных документов в один, например, для оформления трудов конференции, можно использовать класс `combine`. Однако `combine` имеет свои ограничения, в частности, конфликтует с некоторыми полезными стилевыми файлами.

Избежать такого рода трудностей позволит «новинка» — стилевой файл `mmro.sty`, который делает возможным не только сверстать сборник из десятков и даже сотен присланных статей, но и отслеживать процессы корректуры, рецензирования и принятия решений по отдельным статьям.

К числу основных функциональных возможностей стилевого файла `mmro.sty` относятся, например, возможности: включать статью в сборник одной командой `\paper{имя файла}` (при этом она может компилироваться как отдельно, так и в составе сборника — для этого никаких изменений в тексте самой статьи не требуется); в любой момент скомпилировать версию сборника, отобрав в него только статьи с заданным статусом, либо статьи для заданного рецензента; вывода в режиме `draft` дополнительной информаций и статистикй о степени готовности сборника и т. д.

Стилевой файл `mmro.sty` и сопровождающая его документация были разработаны летом 2007 года при подготовке Всероссийской конференции «Математические методы распознавания образов» (ММРО-13). (см. Воронцов К. В. Подготовка сборника трудов конференции в системе LaTeX2e. 2007, 34 с. http://www.ccas.ru/voron/latex.html).

Создание учебно-методических и других полезных материалов

В процессе преподавательской деятельности приходится создавать много разнообразной «мелкой» документации. Для создания программ курсов специально разработаны классы `courseoutline` и `coursepaper`, для печати объявлений о занятиях — класс `assignment`, для вопросников класс `qcm`, кроме того, могут оказаться полезными классы для составления обложек для CD (`cd`), рабочего журнала `labbook` (для биологов), карточек для запоминания иностранных слов (`flashcards`).

Лекция 3

Поддержка письменностей и национальных типографских особенностей

LaTeX и поддержка многоязычной среды. Пакеты babel, ncc, eskdx, gost. Автоматические переносы слов.

В момент возникновения TEX'а его создатель Дональд Кнут, конечно не предполагал, что эта система будет популярна во всем мире. Поэтому без дополнительных усилий TEX и LATEX поддерживают издание только англоязычной типографской продукции. При попытке использовать TEX для верстки текстов на других языках, возникает ряд трудностей.

Во-первых, многие языки содержат буквы, отсутствующие в английском языке. Кнут заложил возможность собирать отдельные буквы некоторых языков (в частности, буквы немецкого языка с умлаутами) из обычных букв и диакритических знаков. Например, чтобы получить Ä и ä, нужно набрать \"{A} и \"{a}.

Во-вторых, для написания текстов на некоторых языках, в том числе, на русском и греческом, необходимо вводить не отдельные новые буквы, а целый алфавит.

В-третьих, в разных языках используются разные правила переноса слов. Использование правил переносов для английского языка даст неверные результаты при обработке немецких, и, тем более, русских текстов. Проблема переносов остается даже при наличии нужного алфавита.

В-четвертых, разные страны имеют разные типографские традиции (использование кавычек, тире, дефиса, формат даты, название разделов документа и прочее). Поэтому сверстанный по англо-американским канонам текст на другом языке кажется странным.

В современном LATEX2e стандартная поддержка набора на любых из десятках языков подключается единообразно, например строки

```
\usepackage[cp1251]{inputenc}
\usepackage[T2A]{fontenc}
\usepackage[russian]{babel}
```

в преамбуле документа, включают поддержку русского языка. Эти пакеты, функции которых станут ясны далее, как раз и позволяют преодолеть указанные выше трудности.

1. Кодировка шрифта в LaTeX

1.1. Кодировка шрифта и кодовая страница

Для того, чтобы понимать и печатать символы различных языков, система и программное обеспечение должна быть способна отличить их от других символов. Поэтому каждый уникальный символ должен иметь уникальное представление внутри операционной системы, или специфического пакета программ. Такая совокупность всех уникальных символов, которые система способна представить сразу, называется *кодировкой (codeset)*.

Кодовая страница (codepage) — это способ хранения информации о шрифте в памяти компьютера, включающий заранее заданный упорядоченный стандартный комплект знаков шрифта для набора текста на определенных языках.

Кодировка шрифта (encoding) представляет собой таблицу символов, где каждой букве алфавита (а также цифрам и специальным знакам) присвоен свой уникальный номер — код символа.

Традиционно языком большинства операционных систем и программного обеспечения является английский, поэтому наиболее распространенной кодировкой была и остается ASCII (Американский Стандартный Код для Информационного Обмена). В расширенной (восьмибитной) ASCII стандартизирована только половина таблицы — 128 символов, которые включают в себя буквы латинского алфавита. Другая же половина таблицы (а всего в ней 256 символов — по количеству состояний, который может принять один байт) отдана под национальные символы, и в каждой стране эта часть различна.

Так как в большинстве систем символы описываются восемью битами, то нет никакого способа расширить ASCII еще больше. Способ создания новых символов в ASCII — это просто создание новых расширенных ASCII реализаций. С помощью этого способа Международная организация

по стандартизации (ISO) определила семейства стандартов, известных как ISO 8859-X. Например, для Европы и Латинской Америки определена кодировка 8859-1 (или, что то же самое, Latin 1), для Восточной Европы — кодировка 8859-2, для кириллицы — кодировка 8859-5, а для Идиш — 8859-8.

Но поскольку стандарт ISO 8859-5 был разработан без принятия во внимание реальных процессов проходящих в России (СССР), он только увеличил количество существующих на тот момент кодировок еще на одну. Вообще вместе с ISO 8859-5 в России существует целых пять различных кодировок. Под словом «различные» подразумевается, что одному и тому же символу соответствует разный цифровой код. Это означает, что если неправильно определить кодировку, то нашему вниманию предстанет абсолютно нечитаемый текст.

Кириллические кодировки появились исторически, в связи с проблемами адаптации к русскому языку разных операционных систем (DOS, MS Windows, UNIX или Mac OS). Кодовые страницы обозначались стандартным индексом или словом.

- Исторически первая и ныне основная в UNIX российская кодировка называется KOI-8.
- Для MS-DOS существует своя DOS-кодировка (кодовая страница 866).
- Стандарт для кодировки Кириллицы в Windows — кодовая страница CP1251. Де-факто, она стала самой распространенной в России.
- Для редких в России пользователей Macintosh-компьютеров была придумана еще одна кодировка — MAC.

В любую кириллическую кодировку обычно входят строчные и прописные знаки латинского и русского алфавита, дополнительные строчные и прописные знаки национальных алфавитов (белорусского, македонского, сербского, украинского), цифры, знаки препинания и другие наборные знаки.

Имеются также другие более гибкие стандарты, отличные от ASCI. Наиболее известен стандарт UNICODE — международный стандарт кодирования знаков. Теоретически он позволяет разместить более миллиона различных знаков, что включает в себя практически все мировые алфавиты и другие системы письма, имеющие значение для полиграфии. Но до момента, когда все перейдут на UNICODE, еще далеко.

1.2. Кодировка входного файла в LaTeX

Поддержка русского языка для LaTeXвключена в MiKTeX. Для создания русскоязычных документов достаточно подключить несколько основных пакетов.

Если во входном файле встречаются символы, не входящие в стандартные 128 символов ASCII, то LaTeX'у необходимо явно указать используемую кодировку. Для восприятия LaTeX'ом кодировки служит пакет `inputenc` — он предназначен для указания кодовой страницы исходного файла. Указанный пакет подключается в преамбуле сразу после объявления класса документа:

```
\usepackage[имя_кодировки]{inputenc}
```

Если файл создается в среде MS Windows, и используется кодовая страница CP1251, то **имякодировки** задается как `cp1251`. В этом случае надо писать `\usepackage[cp1251]{inputenc}`. В MS-DOS со стандартной русской кодовой страницей 866 вместо **имякодировки** надо указать `cp866`. В системах, основанных на UNIX с кодовой таблицей KOI-8 **имякодировки** задается как `koi8-r` и `koi8-u`.

В LaTeX'е предусмотрена возможность работать с файлом, содержащим символы в разных кодировках. В этом случае текущую кодировку можно объявлять прямо в документе посредством декларации:

```
\inputencoding{имя_кодировки}
```

Этот пакет очень важен для русскоязычных пользователей, ведь именно он (в паре с `fontenc`) дает возможность одинаково успешно пропускать через компилятор LaTeX файлы, подготовленные в различных операционных системах и при различных установках кодовой страницы.

	´0	´1	´2	´3	´4	´5	´6	´7	
´00x	0 Ѣ	1 Ѣ	2 Ц	3 Э	4 І	5 Є	6 Ѣ	7 Ѣ	˝0x
´01x	8 ѣ	9 ѣ	10 ц	11 э	12 і	13 є	14 ѣ	15 ѣ	
´02x	16 Ю	17 Ж	18 Й	19 Ё	20 V	21 ~~Ѳ~~	22 S	23 Я	˝1x
´03x	24 ю	25 ж	26 й	27 ё	28 v	29 ~~ѳ~~	30 s	31 я	
´04x	32 ¨	33 !	34 ”	35 Ѣ	36 ˘	37 %	38 ´	39 ’	˝2x
´05x	40 (	41)	42 *	43 Ѣ	44 ,	45 -	46 .	47 /	
´06x	48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7	˝3x
´07x	56 8	57 9	58 :	59 ;	60 «	61 ı	62 »	63 ?	
´10x	64 ˇ	65 А	66 Б	67 Ц	68 Д	69 Е	70 Ф	71 Г	˝4x
´11x	72 Х	73 И	74 J	75 К	76 Л	77 М	78 Н	79 О	
´12x	80 П	81 Ч	82 Р	83 С	84 Т	85 У	86 В	87 Ш	˝5x
´13x	88 Ш	89 Ы	90 З	91 [	92 “	93]	94 Ь	95 Ъ	
´14x	96 ‘	97 а	98 б	99 ц	100 д	101 е	102 ф	103 г	˝6x
´15x	104 х	105 и	106 j	107 к	108 л	109 м	110 н	111 о	
´16x	112 п	113 ч	114 р	115 с	116 т	117 у	118 в	119 щ	˝7x
´17x	120 ш	121 ы	122 з	123 –	124 —	125 №	126 ь	127 ъ	
	˝8	˝9	˝A	˝B	˝C	˝D	˝E	˝F	

Рис.3.1. Пример кодировочной таблицы TEX’а для шрифта wncyr10

	´0	´1	´2	´3	´4	´5	´6	´7	
´00x	0 Ѐ	1 Ё	2 Ѣ	3 Ѓ	4 Є	5 S	6 І	7 Ї	˝0x
´01x	8 J	9 Ѣ	10 Ѣ	11 Ѣ	12 Ѓ	13 Й	14 Ў	15 Ц	
´02x	16 А	17 Б	18 В	19 Г	20 Д	21 Е	22 Ж	23 З	˝1x
´03x	24 И	25 Й	26 К	27 Л	28 М	29 Н	30 О	31 П	
´04x	32 Р	33 С	34 Т	35 У	36 Ф	37 Х	38 Ц	39 Ч	˝2x
´05x	40 Ш	41 Щ	42 Ъ	43 Ы	44 Ь	45 Э	46 Ю	47 Я	
´06x	48 а	49 б	50 в	51 г	52 д	53 е	54 ж	55 з	˝3x
´07x	56 и	57 й	58 к	59 л	60 м	61 н	62 о	63 п	
´10x	64 р	65 с	66 т	67 у	68 ф	69 х	70 ц	71 ч	˝4x
´11x	72 ш	73 щ	74 ъ	75 ы	76 ь	77 э	78 ю	79 я	
´12x	80 ѐ	81 ё	82 ѣ	83 ѓ	84 є	85 s	86 і	87 ї	˝5x
´13x	88 j	89 ѣ	90 ѣ	91 ѣ	92 ѓ	93 й	94 ў	95 ц	
´14x	96 Ѡ	97 ѡ	98 Ѣ	99 ѣ	100 Ю	101 ю	102 Ѧ	103 ѧ	˝6x
´15x	104 Ѩ	105 ѩ	106 Ѫ	107 ѫ	108 Ѭ	109 ѭ	110 Ѯ	111 ѯ	
´16x	112 Ѱ	113 ѱ	114 Ѳ	115 ѳ	116 V	117 v	118 Ѵ	119 ѵ	˝7x
´17x	120 Ѷ	121 ѷ	122 Ѻ	123 ѻ	124 Ѽ	125 ѽ	126 Ѿ	127 ѿ	
´20x	128 Ҁ	129 ҁ	130 ҂	131 ҃	132 ҄	133 ҅	134 ҆	135	˝8x
´21x	136 ҇	137 ҇	138 Ҋ	139 ҋ	140 Ѣ	141 ѣ	142 Ҍ	143 ҍ	
´22x	144 Ҏ	145 ҏ	146 Ґ	147 ґ	148 Ґ	149 ґ	150 Җ	151 җ	˝9x
´23x	152 Ҙ	153 ҙ	154 Ҝ	155 ҝ	156 Ҝ	157 ҝ	158 Ҟ	159 ҟ	
´24x	160 Ҡ	161 ҡ	162 Ҥ	163 ҥ	164 Ҥ	165 ҥ	166 Ҧ	167 ҧ	˝Ax
´25x	168 Ҩ	169 ҩ	170 Ҫ	171 ҫ	172 Ҭ	173 ҭ	174 Ү	175 ү	
´26x	176 Ұ	177 ұ	178 Ҳ	179 ҳ	180 Ҵ	181 ҵ	182 Ҷ	183 ҷ	˝Bx
´27x	184 Ҹ	185 ҹ	186 Ҽ	187 ҽ	188 Ҿ	189 ҿ	190 Ҿ	191 ҿ	
´30x	192 І	193 Ж	194 ж	195 Ҕ	196 ҕ	197 Ҕ	198 ҕ	199 Ҕ	˝Cx
´31x	200 ҕ	201 Ҕ	202 ҕ	203 Ҕ	204 ҕ	205 Ҕ	206 ҕ	207	
´32x	208 Ӑ	209 ӑ	210 Ӓ	211 ӓ	212 Ӕ	213 ӕ	214 Ӗ	215 ӗ	˝Dx
´33x	216 Ә	217 ә	218 Ӛ	219 ӛ	220 Ж	221 ж	222 Ӝ	223 ӝ	
´34x	224 З	225 з	226 Ӣ	227 ӣ	228 Ӣ	229 ӣ	230 Ӧ	231 ӧ	˝Ex
´35x	232 Ѳ	233 ѳ	234 Ӧ	235 ӧ	236 Ӭ	237 ӭ	238 Ӯ	239 ӯ	
´36x	240 Ӱ	241 ӱ	242 Ӳ	243 ӳ	244 Ӵ	245 ӵ	246	247	˝Fx
´37x	248 Ӹ	249 ӹ	250	251	252	253	254	255	
	˝8	˝9	˝A	˝B	˝C	˝D	˝E	˝F	

Рис.3.2. Пример кодировочной таблицы TEX’а для шрифта ucr10

1.3. Внутренняя кодировка шрифта в LaTeX

Выше говорилось, что за перевод из кодировки файла во внутреннюю кодировку LaTeXотвечает пакет `inputenc`. В качестве опции при загрузке с ним передаётся текущая восьмибитная кодовая страница документа. Далее LaTeX'у необходимо сообщить, какую внутреннюю кодировку шрифта следует использовать в самом документе.

По умолчанию LaTeXиспользует 7-битную кодировку текстовых шрифтов OT1, в которой нет русских букв. Кроме неё для LaTeX'а разработана кодировка T1, включающая 256 символов и содержащая большинство букв языков на латинской графической основе.

По той же причине, по которой появилось несколько кодировок ISO 8859-X, в LaTeX'е есть несколько стандартизованных 8-битных кодировок, включающих русские и другие кириллические буквы (T2A, T2B, T2C и X2), а также не стандартизованные варианты. Они охватывают практически все языки, основанные на кириллице. Кодировки T2A, T2B и T2C включают в себя буквы латинского языка и поэтому подходят для смешанных текстов. Для документов на русском языке обычно используется кодировка T2A, хотя буквы русского алфавита содержатся во всех кириллических кодировках. Кодировки T2B и T2C отличаются наборами редких символов, например некоторые старославянские буквы есть только в кодировке T2C, а если основным языком документа является хакасский, то придется использовать кодировку T2B (см. примеры кодовых таблиц на рисунках 3.1, 3.2).

Узнать, в какой кодировке определены буквы различных национальных алфавитов, можно в руководстве `cyrguide`.

Необходимая кодировка текстовых шрифтов указывается посредством пакета `fontenc`:

```
\usepackage[кодировка шрифта]{fontenc}
```

Например, если язык итогового документа русский, обычно указывают

```
\usepackage[T2A]{fontenc}
```

Можно указать сразу несколько кодировок через запятую. Заметим, что кодировкой по умолчанию станет та из них, которая указана последней.

Если необходимо поменять кодировку шрифта в теле документа, используются следующие декларации:

```
\fontencoding{кодировка шрифта}
\selectfont
```

При этом команда `\fontencoding` только объявляет кодировку, а команда `\selectfont` служит для того, чтобы LaTeXстал использовать шрифты в объявленной кодировке. Между этими командами не должно быть текста (допускаются только пробелы и декларации выбора других атрибутов шрифтов). Кодировка шрифта должна быть стандартной, иначе ее надо объявить в преамбуле посредством `fontenc`.

2. Поддержка переносов в LaTeX. Двуязычные документы

Основным языком документа считается тот, на котором печатается предопределенный текст, например, в английском документе подпись к таблице начинается со слова Table, а не со слова Таблица, как должно быть в русском документе.

Для русских документов кодировка текстовых шрифтов T2A по умолчанию устанавливается при подключении пакета `russianb`. Он является частью системы многоязыковой поддержки `babel` и загружается следующим образом:

```
\usepackage[russian]{babel}
```

Пакет `russianb` не только устанавливает кодировку, но и включает правила переноса русских слов. Заметим, что образцы допустимых переносов в словах русского языка подключаются на стадии создания файла формата. Если в файл формата не включены правила переноса на русском языке, то LaTeX будет верстать документы без переносов в словах.

Русскоязычный документ может содержать отдельные английские слова и даже целые абзацы английского текста. В последнем случае возникнет

проблема корректных переносов слов английского языка. Чтобы включить соответствующие правила переноса, перед англоязычным абзацем ставится любая из эквивалентных команд

```
\English   или   \Eng
```

После окончания английского текста для восстановления правил переноса в русском языке надо поставить любую из двух эквивалентных команд

```
\Russian   или   \Rus
```

Иногда возникает необходимость включить в англоязычный текст небольшие фрагменты текста на русском языке. Если для автора не имеет значение наличие длинных или разреженных строк в тексте, то можно ограничится подключением пакета `fontenc` с указанием кодировки T2A. В этом случае LaTeX сверстает русский текст без переносов в словах.

Для небольшого количества русского текста места переноса в русских словах можно задать командой \– или в декларации `\hyphenation`.

Если документ на английском языке содержит большие фрагменты русского текста (или наоборот, русский тескт большие фрагменты на английском), то лучше пользоваться пакетом `babel` с двумя опциями:

```
\usepackage[russian,english]{babel}
\usepackage[english,russian]{babel}
```

Если опция `english` стоит последней, то основным языком документа будет английский (и наоборот). В случае англоязычного текста перед абзацами на русском языке ставится команда `\Russian` (или `\Rus`), а после них — `\English` (или `\Eng`).

Для вставки коротких фраз на русском языке можно пользоваться командой `\textcyrillic{русский текст}`.

Переключение алгоритмов переносов слов с одного языка на другой также осуществляется командой `\selectlanguage`, в аргументе которой указывается английское название языка (например, `\selectlanguage{russian}`). Она может даваться в преамбуле или в теле документа.

3. Особенности русскоязычного набора

3.1. Кириллица и латиница

Русскоязычная статья не должна содержать терминологии на иностранных языках, если существует общепринятый перевод терминов на русский язык.

Фамилии и инициалы иностранных авторов указываются в русской транскрипции. В естественнонаучной среде приняты Британо-американские правила транслитерации:

А	A	З	Z	П	P	Ч	CH
Б	B	И	I	Р	R	Ш	SH
В	V	Й	I	С	S	Щ	SHCH
Г	G	К	K	Т	T	Ы	Y
Д	D	Л	L	У	U	Ь	’
Е	E	М	M	Ф	F	Э	É
Ё	E	Н	N	Х	KH	Ю	YU
Ж	ZH	О	O	Ц	TS	Я	YA

Использование букв русского алфавита в формулах допускается в исключительных случаях: когда это является общепринятым в конкретном научном направлении и оформляется принятым способом.

Пакет `russianb`, кроме всего прочего, объявляет *кириллические математические алфавиты*. Кириллические буквы в математической моде можно использовать после загрузки пакета `mathtext` из T2 но перед загрузкой пакета `babel` с опцией `russian`.

В отличие от латинских букв по умолчанию буквы имеют прямое начертание. Если нужен шрифт с иным начертанием, то надо воспользоваться кириллическими аналогами стандартных математических алфавитов:

```
\cyrmathrm \cyrmathbf \cyrmathsf \cyrmathit \cyrmathtt
```

Заметим, что включить в формулы русский текст можно без подключения указанных шрифтов, воспользовавшись командой `\text{русский текст}` из пакета `amstext`.

Пакет `russianb` также дает возможность печатать имена функций (тригонометрических, гиперболических и др.) согласно российской традиции (в скобках указаны названия функций):

```
\arctg  (arctg) \ch     (ch)  \ctg (ctg)   \sh (sh)
\arcctg (arcctg)\cosec (cosec)\cth (cth)   \tg (tg)
\th     (th)    \Prob    (P)  \nod (н.о.д.)\nok(н.о.к.)
\Proj   (Пр)    \Variance(D)  \NOD (НОД)   \NOK(НОК)
```

Команды `\nod`, `\nok`, `\NOD`, `\NOK` и `\Proj` работают только после подключения пакета `mathtext`.

3.2. Сокращения и пробелы в русском тексте

Правила сокращений слов и словосочетаний устанавливает ГОСТ 7.12-93. Не допускаются сокращения: т. о. (таким образом); т. н. (так называемый); т. к. (так как). Без расшифровки используются только сокращения, заведомо понятные читателю: ЭВМ, КПД, ГЭС и т. п. Другие сокращения должны быть расшифрованы при первом упоминании. Форма сокращений по всей работе должна быть одинакова.

Буквенные аббревиатуры — это сокращения, образованные из первых букв слов, входящих в словосочетание, и произносимые при чтении в сокращенной форме: МХАТ, вуз, ЭВМ, КЗоТ, — склоняются, если читаются по слогам, а не по буквам и род ведущего слова совпадает с родовой формой самой аббревиатуры. При этом падежное окончание пишется строчными буквами и слитно: МХАТа, вуза, КЗоТа. Сложносокращенные слова (ГОСТ, ОСТ) не склоняются!

При стечении в конце сокращения двух одинаковых согласных одна усекается: ил., а не илл.

Между словами внутри абзаца ставится один пробел; перед знаком препинания, закрывающей скобкой или кавычкой пробелы не ставятся, после знака препинания, кавычки или закрывающей скобки ставится один пробел. Перед открывающей скобкой, кавычкой и длинным тире ставится один пробел, после них пробел не ставится.

При выравнивании текста в абзаце пробел между предложениями может увеличиться сильнее, чем между словами. Это делается автоматически и облегчает восприятие предложений в тексте. При этом точка воспринимается как признак конца предложения. С другой стороны, пробелы после предлога, относящегося к последующему слову и после точки, не являющейся концом предложения, не должны при этом увеличиваться. Поэтому после точек, не являющихся концом предложения, нужно ставить жесткий пробел «\» или неразрывный жесткий пробел «~».

Рекомендуемая для набора в LaTeX2e запись «О.~Т.~Личник» и в середине предложения «и~т.~д.\» обеспечивает нерастяжимые неразрывные и разрывный пробелы нужной величины.

Между последней цифрой числа и буквенным обозначением единицы измерения следует вставлять неразрывный пробел 352~МПа, 35~cm, однако между числом и обозначениями градусов, минуты, секунд, процентом пробел не ставится `$30^\circ$C`, `$10\%$`.

3.3. Кавычки, дефисы, тире, управление переносами слов

В английском языке приняты одинарные и двойные кавычки в виде ‘...’ и “...”. *В России* приняты французские кавычки «елочки» и немецкие кавычки „лапки“. „Лапки“ обычно используются внутри «ёлочек», например, «...наш гордый „Варяг“...». После подключения пакета `russianb` в русском тексте возможны следующие способы набора:

- кавычки-лапки „...“ набираются "“...”" или `\glqq` ... `\grqq`;
- кавычки-елочки «...» набираются "<...">" или `\flqq` ... `\frqq`, или <<...>>.

Активный символ " вообще не является кавычками, поэтому писать "..." — неправильно!!!

Для печати *дефиса* в составных словах введены следующие команды:

" ~ печатает дефис «-» и запрещает делать переносы в самих словах;

"= печатает дефис «-», TeX как может сам делает переносы в словах;

"— задает место в слове, где можно делать перенос, не запрещяя переносы и в других местах слова (в отличие от команды `\-`);

" " задает место в слове, где можно делать перенос, причем дефис при переносе в этом месте не ставится;

" , вставляет небольшой пробел после инициалов с правом переноса в фамилии.

Пакет `russianb` вводит три команды для набора тире, которые печатаются согласно российской традиции:

"--- используется для печати тире «— » в тексте. Оно несколько короче английского длинного тире. Кроме того, команда задаёт небольшую жёсткую отбивку от слова, стоящего перед тире. При этом, само тире не отрывается от слова. После тире следует такая же отбивка от текста, как и перед тире. При наборе текста между словом и командой, за которым она следует, должен стоять пробел.

"--~ используется для печати составных слов, таких, как «Закон Менделеева—Клапейрона». Она ставит более короткое, по сравнению с английским, тире и позволяет делать переносы во втором слове. При наборе текста эта команда не отделяется пробелом от слова, за которым она следует (Менделеева"--~). Следующее за командой слово может быть отделено от неё пробелом или перенесено на другую строку.

"--* служит для печати тире, перед прямой речью, начинающейся с абзаца. Она печатает русское тире и жёсткую отбивку нужной величины перед текстом.

Вместо команд, начинающихся с символа ", можно использовать аналогичные им по действию, но более длинные в наборе, команды `\cdash---`, `\cdash--~` и `\cdash--*`, соответственно.

Чтобы получить автоматические переносы в составных словах и русское тире, соответствующее к тому же нормам и традициям российского книгопечатания, рекомендуется воспользоваться в преамбуле командой

```
\usepackage[shortcuts,cyremdash]{extdash}
```

При этом набор составных слов слегка отличается от описанного для `russianb`, и нужно различать в наборе следующие случаи:

`\=/` составное слово можно переносить по слогам, но не по дефису: `$\sigma$\=/конечная` и `$L$\=/аппроксимация`;

`\-/` составное слово можно переносить как по слогам, так и по дефису: `розово\-/фиолетовый`, `Римский\-/Корсаков` (один человек);

`\--` связанные коротким тире слова можно переносить как по слогам, так и по дефису: закон `Бойля\--Мариотта` (два ученых), `десять\--двенадцать этажей`;

`\---` длинное тире, связывающее предложения или сочетания слов: `и все это \--- бесплатно`.

При этом продолжают действовать указания на возможный перенос: «`\-`», блокирующий автоопределение других возможностей переноса в данном слове, служит для явного указания всех разбиений на слоги; «`"-` », разрешающий автоматический перенос по слогам конца слова, служит для коррекции переносов в начале слова.

3.4. Команды с предопределённым текстом

Пакет `russianb` переопределяет команду `\today` так, чтобы она печатала названия месяцев по русски. Например, 1 ноября 2008 г. Кроме того, он переопределяет также команды из стандартных классов и некоторых пакетов, которые печатают предопределённый текст.

Они приведены ниже вместе с английским и русским текстом:

`\abstractname`	Abstract	Аннотация
`\appendixname`	Appendix	Приложение
`\bibname`	Bibliography	Литература
`\chaptername`	Chapter	Глава
`\contentsname`	Contents	Содержание (в классе article) Оглавление (в классах book и report)
`\figurename`	Figure	Рис.
`\indexname`	Index	Предметный указатель
`\listfigurename`	List of Figures	Список иллюстраций
`\listtablename`	List of Tables	Список таблиц
`\partname`	Part	Часть
`\refname`	References	Список литературы
`\tablename`	Table	Таблица
`\seename`	see	см. (пакет `makeidx`)
`\alsoname`	see also	см. также (пакет `makeidx`)
`\proofname`	Proof	Доказательство (`amsthm`)

Определены также две новые команды: `\prefacename` (Предисловие) и `\authorname` (Именной указатель).

Кроме того, существует возможность переопределить команды посредством декларации `\renewcommand`. Например, после переопределения в теле документа

```
\renewcommand{\indexname}{Алфавитный указатель}
```

перед указателем терминов будет напечатано «Алфавитный указатель», а не «Предметный указатель».

Лекция 4 ⏮

Управление шрифтовыми выделениями и подключение шрифтов

Характеристики шрифтов. Стандартные пакеты, пакеты cmsuper, LH, pscyr, urwcyr.

1. Классификация наборных шрифтов (Typeface classification)

Шрифт — это система визуального отображения информации при помощи условных символов. В более узком значении это комплект знаков определенного рисунка — литер, цифр и специальных знаков. Существуют стандарты, в которых регулируются технологические параметры шрифтов.

Большое разнообразие шрифтов, созданных к началу XX в., привело к необходимости разработать классификацию шрифтов. В России классификация шрифтов в основном построена применительно к требованиям издательств и типографий. Типографские алфавиты шрифтов принято называть «гарнитурой». К основным признакам шрифта можно отнести характер рисунка символов, начертание очка и размер.

Гарнитура — это совокупность шрифтов, объединенных одним характером рисунка букв, вне зависимости от их разновидностей и размеров. Каждая гарнитура получила свое название, например, литературная, обыкновенная, академическая и т. д.

В зависимости от принципов построения и важнейших графических признаков шрифты принято делить по рисунку на несколько основных и дополнительных групп, отличающихся наличием засечек, их размером и формой, а также контрастностью. В каждую группу может входить несколько гарнитур. Например, по своим графическим признакам русский гражданский шрифт сходен и может быть отнесен в одну группу со шрифтами типа антиква но сильно отличается от готических шрифтов.

Все гарнитуры можно подразделить на четыре группы, три основных и одну дополнительную.

- Шрифты с засечками — шрифты группы Антиква (Antiqua, Serif typefaces).

Для них характерна ярко выраженная контрастность вертикальных и горизонтальных штрихов. Эта группа делится на старую и новую Антиквы.

К старой Антикве относятся гарнитуры с умеренной контрастностью штрихов (1:3), с засечками в виде плавного утолщения концов основных штрихов, чаще всего приближающихся к треугольнику, преимущественно с наклонными осями округлых букв. Данная группа шрифтов сложилась еще в античном мире. Из старых русских образцов в эту группу вошли петровская гражданская азбука и шрифты типографии Московского университета конца XVIII в. Из современных образцов к этой группе относятся Литературная, Банниковская, Лазурская и др.

Это шрифт гарнитуры Антиква!
Это шрифт гарнитуры Лазурская!

К новой антикве относятся гарнитуры с контрастными штрихами (1:5), с длинными тонкими горизонтальными засечками, соединяющимися с основными штрихами под прямым углом (иногда с легким закруглением). Округлые буквы имеют вертикальные оси. В России эта группа шрифтов сложилась в середине XVIII в. К ней относятся такие гарнитуры, как Елизаветинская, Обыкновенная и др.

К новой антикве также относятся группа гарнитур, имеющих малоконтрастные штрихи (1:5) с длинными засечками, преимущественно с закругленными концами, соединенными с основными штрихами под прямым углом или с легким закруглением. Рисунки этих шрифтов возникли в конце XIX в. К этой группе относятся такие гарнитуры, как Школьная, Журнальная, Пискаревская, Новая журнальная и др.

Это шрифт гарнитуры журнальная!

- Группа брусковых шрифтов.

К этой группе относятся гарнитуры с малоконтрастными штрихами с длинными засечками, соединенными с основными штрихами под прямым углом

или с легким закруглением. Брусковые шрифты отличаются тем, что каждая буква и цифра состоит из штрихов, представляющих собой прямые линии. Засечки имеют толщину, равную примерно половине толщины основного штриха. Шрифты этой группы начали применятся в начале XIX в. на рекламных объявлениях, в газетных заголовках и т. д. К брусковым шрифтам относятся шрифты: египетский, итальянский. Гарнитуры имеют названия: Брусковая, Балтика, Реклама.

Это шрифт гарнитуры Балтика!

- Группа рубленых (гротескных, плакатных) шрифтов (Sans Serif, Gothic).

Собирательное название шрифтов без засечек. В Англии и Франции они чаще называются Sans Serif, в Америке Gothic, в Германии Grotesk. Эту группу шрифтов характеризует отсутствие засечек и одинаковая толщина всех штрихов (с контрастностью 1:1). Они начали широко применяться в начале XIX в. в заголовках рекламных объявлений. Сюда относят плакатные гарнитуры Журнальная рубленая, Газетная рубленая, Ариал, Тахома, Букварная и др.

Это шрифт гарнитуры Журнальная рубленая!

Это шрифт гарнитуры Ариал!

- Дополнительная группа.

К ней относятся все шрифты, которые не могут быть включены в указанные выше группы, например, рукописные, различные каллиграфические шрифты, рисунок который выполнен с помощью ширококонечного пера (рондо) или кисти. Шрифты дополнительной группы не имеют общих графических признаков.

Это шрифт гарнитуры Рекламный!

Помимо основных графических признаков, для характеристики различных гарнитур и их распознавания можно пользоваться дополнительными

признаками — характерными (типовыми) особенностями в рисунке отдельных букв: форма точек (завитков) у букв; расположение и конфигурация засечек, нижних и верхних выносных элементов.

Одна гарнитура внутри группы объединяет шрифты, различные по начертанию (*курсив*, **жирное**, полужирное) и размерам (кеглю), но с одинаковым рисунком.

При наборе текста заглавия, сноски и некоторые другие его элементы принято набирать шрифтом, отличающимся по размерам от основного шрифта текста. Шрифты одной гарнитуры классифицируются по размеру в соответствии с величиной, характеризующей размер шрифта; в полиграфии она называется кегль или кегель (Size, Body size, Point size).

Практически **кегль** — это высота буквы или знака в строке вместе со свободным пространством над и под ним, необходимым для того, чтобы штрихи букв в смежных строках не сливались друг с другом. Кегль измеряется в пунктах (1 pt = 0,376 мм). Например, высота верхней площадки литеры шрифта кегля 12 равна 12 pt, высота строчной буквы занимает меньше половины кегля.

Почти все кегли имеют собственные названия: петит (8 pt), миттель (14 pt), текст (20 pt), малый канон (32 pt) и др.

Шрифты одной гарнитуры по начертанию очка (по наклону основных штрихов) подразделяются на шрифты прямого наклонного и курсивного начертаний.

У шрифтов прямого начертания основные штрихи расположены вертикально. У курсивных и наклонных начертаний шрифтов основные штрихи имеют наклон (вправо или влево, примерно на 15°). Разница между курсивным и наклонным начертаниями шрифта состоит в том, что *строчные буквы шрифта курсивного начертания строятся по принципу рукопис-*

ных шрифтов, а *строчные буквы шрифта наклонного начертания имеют сходство с буквами прямого начертания.*

2. Выбор шрифтов и их атрибутов в LaTeX

Несмотря на огромное количество шрифтов, созданных для компьютерных издательских систем, все их можно разделить на четыре группы: шрифты с засечками (антиква — serif), шрифты без засечек (гротески — sans serif), декоративные шрифты (decorative), рукописные шрифты (script).

В LaTeX'е доступны разные виды текстовых шрифтов, которые характеризуется следующими четырьмя атрибутами: семейством (`family`); насыщенностью (`series`); начертанием (`shape`); размером (`size`).

Семейство шрифта выражает примерно то же самое, что и группы гарнитур (с засечками, без засечек и т. д.). В стандартной комплектации LaTeXопределены три семейства: шрифты с засечками, шрифты без засечек и `шрифты типа «пишущая машинка»`.

Насыщенность определяет ширину и жирность шрифта. В стандартной комплектации LaTeXвозможны два варианта насыщенности: средняя и **полужирная**.

Кроме того, предусматривается четыре варианта начертания: прямое начертание, *курсивное начертание*, *наклонное начертание* и НАЧЕРТАНИЕ «КАПИТЕЛЬ».

Семейство, насыщенность, начертание и размер шрифта могут различным образом сочетаться. Каждый из шрифтовых атрибутов можно менять независимо от остальных.

2.1. Команды для изменения атрибутов шрифта

Команды для изменения атрибутов шрифта существуют в LaTeX'е в двух вариантах: *в виде команды без аргумента и команды с аргументом*.

Команда без аргумента меняет атрибут текущего шрифта вплоть до того момента, пока он не будет изменен другой командой. Если эта команда дана внутри группы, то она определяет атрибут шрифта пока не закончится группа.

Команда с одним аргументом, помещаемым в фигурные скобки, меняет атрибут шрифта только у своего аргумента.

Для первых трех атрибутов шрифта (кроме размера) имя команды без аргумента совпадает с английским названием соответствующего атрибута:

`\rmfamily` — шрифты с засечками;
`\sffamily` — шрифты без засечек;
`\ttfamily` — шрифты типа «печатная машинка»;

`\mdseries` — средняя насыщенность;
`\bfseries` — полужирный;

`\upshape` — прямое начертание;
`\itshape` — курсив;
`\slshape` — наклонное начертание;
`\scshape` — капитель.

Имя команды с аргументом состоит из слова `text`, к которому добавлены две буквы, описывающие атрибут, например, `\textbf{...}` или `\textsc{...}`.

Теперь мы можем сделать вид, что **печатаем на печатной машинке**, затем шрифт с засечками заменить на **шрифт без засечек**, после чего **подчеркнуть важность этого текста** *курсивным* или **полужирным шрифтом** и, наконец попробовать ***соединить три в одном***.

При наборе этой фразы использовались команды как с аргументом, так и без аргумента в разных сочетаниях:

```
Теперь мы можем сделать вид, что
\texttt{печатаем на печатной машинке},
затем шрифт с засечками заменить на
\textsf{шрифт без засечек}, после чего
\sffamily подчеркнуть важность этого текста
\textit{курсивным} или \textbf{полужирным шрифтом}
\textnormal{и, наконец попробовать}
\textbf{\textit{соединить три в одном}}.
```

Специальная команда `\emph{...}` позволяет набирать выделяемую фразу курсивом, если текущий шрифт прямой и и прямым шрифтом, если текущий шрифт наклонный. В обычном тексте, набранном прямым шрифтом, результаты действия команд `\emph{...}` и `\textit{...}` неразличимы.

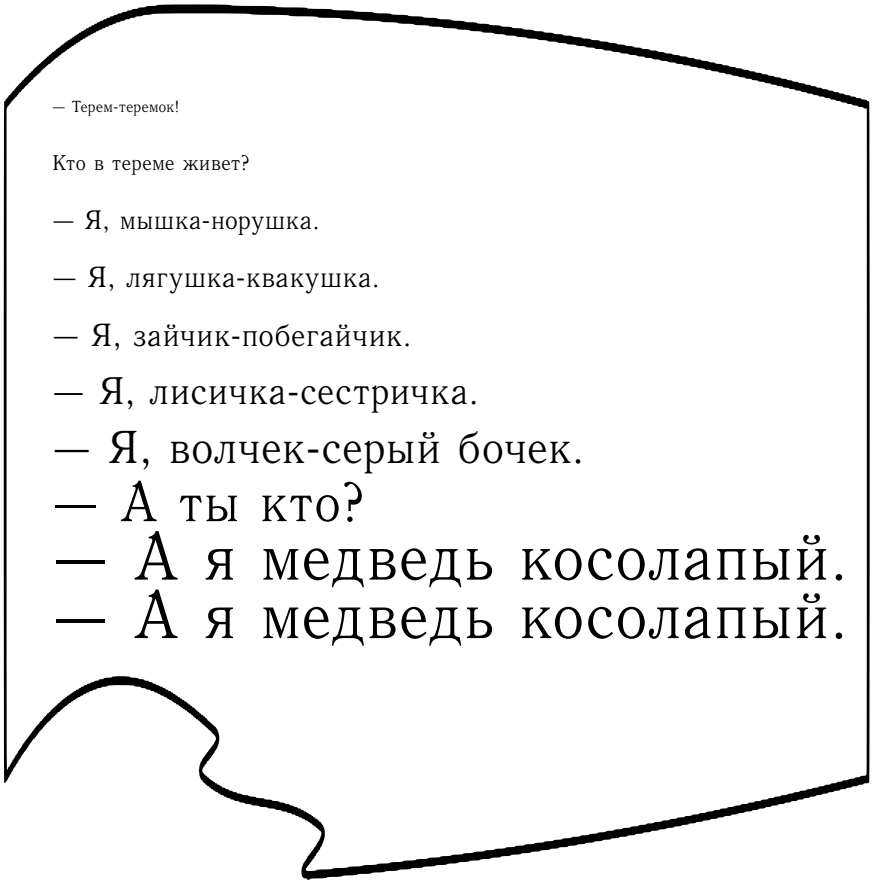
Обратим внимание на то, что команда `\textnormal` сразу возвращает к изначальному шрифту, который был по умолчанию. Вариант этой команды без аргумента `\normalfont` работает также, т.е. переключает шрифт на основной шрифт документа.

Изменение размера шрифта определяется командами без аргумента. Всего их десять и все они использованы в следующем примере.

```
{\tiny          -- Терем-теремок! }        \\
{\scriptsize      Кто в тереме живет?}    \\
{\footnotesize -- Я, мышка-норушка.}      \\
{\small        -- Я, лягушка-квакушка.}   \\
{\normalsize   -- Я, зайчик-побегайчик.}  \\
{\large        -- Я, лисичка-сестричка. } \\
{\Large        -- Я, волчек-серый бочек.} \\
{\LARGE        -- А ты кто?}              \\
{\huge         -- А я медведь косолапый.} \\
{\Huge         -- А я медведь косолапый.}
```

Кегль шрифтов, задаваемых командами `\tiny` и т. д., на самом деле не постоянный, а зависит от класса документа и классовых опций. В стан-

дартных классах с основным шрифтом кегля 12 команды `\huge` и `\Huge` задают один и тот же размер (кегль 25). При компиляции нашего примера получится:



Размер шрифта `\scriptsize` используется для изображения индексов, размер `\footnotesize` используется в сносках, а шрифты размера больше, чем `\normalsize` — для набора заголовков.

2.2. Добавление одного шрифта

Иногда в тексте приходится несколько раз использовать какой-нибудь особый символ или экзотический шрифт для выделения отдельных частей текста. Простой способ добавить в текст отдельный шрифт (если соответствующий файлы находятся в тех местах, где TEXих ищет) предоставляет команда `\newfont`. Она имеет формат

```
\newfont{\команда}{описание_шрифта},
```

где `команда` — это придуманная команда для переключения на добавляемый шрифт, а `описание шрифта` — это имя `tfm`-файла, соответствую-

щего данному шрифту.

Например, чтобы использовать в документе английский причудливый шрифт, разработанный Дональдом Кнутом, нужно знать только его внутреннее имя `cmff10` (файлы `cmff10.tfm` и `cmff10.mf` входят в стандартный комплект шрифтов LaTeX). Чтобы пользоваться этим шрифтом в своем тексте, в его преамбуле или любом другом месте дается команда `\newfont{\strannii}{cmff10}`, после чего шрифт можно подключить так:

```
Strange font: {\strannii strange, strange, strange}
```

После компиляции получим:

Strange font: strange, strange, strange

С помощью команды, определенной в `\newfont`, можно переключаться на новый шрифт внутри группы. В отличие от команд `\itshape` или `\bfseries` при ее использовании всегда подключается шрифт только того размера, что записан в `tfm`-файле, на который ссылается ее второй аргумент. Внутри математической формулы такая команда вообще будет проигнорирована.

Описанным выше способом можно вывести на печать отдельный символ. Например, во входящем в комплект MikTeX'а шрифте `wasy10` в позиции 45 имеется символическое изображение черного смайлика. Зная это, можно подключить шрифт, дав команду `\newfont{\wasyten}{wasy10}`, определить команду `\bs` при помощи `\newcommand{\bs}{\wasyten\symbol{45}}` и использовать ее для изображения смайлика ☻.

С помощью команды `\newfont` можно подключить масштабированный шрифт. Это возможно сделать двумя способами.

Можно задать требуемое увеличение или уменьшение во втором аргументе команды `\newfont` с помощью слова `scaled`, за которым следует целое число, выражающее коэффициент масштабирования, умноженный на 1000. Например, чтобы увеличить символ смайлика в два с половиной раза, надо после имени `tfm`-файла написать `scaled 2500`.

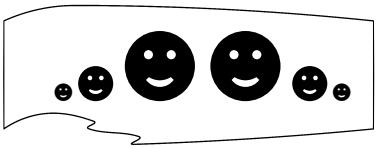
Можно также задать увеличение не в явном виде, а сообщить TEX'у требуемый «характерный размер» шрифта. Для этого надо во втором аргументе команды `\newfont` написать после имени `tfm`-файла масштабируемого шрифта `at размер`, где `размер` — требуемый размер, заданный обычным образом в TEX'овских единицах длины или через TEX'овские параметры длины. Если основной шрифт документа имеет кегль 10, то «характерный размер» разумно (но необязательно) выбирать равным 10pt, если 11 или 12, то 11pt или 12pt.

Рассмотрим работу указанных команд на примере изображения смайлика:

```
\newfont{\wasy}{wasy10}
\newfont{\bigwasy}{wasy10 scaled 2000}
\newfont{\largewasy}{wasy10 at 40pt}
\newcommand{\bs}{\wasy\symbol{45}}
\newcommand{\bbs}{\bigwasy\symbol{45}}
\newcommand{\lbs}{\largewasy\symbol{45}}

\bs \bbs \lbs \lbs \bbs \bs
```

После компиляции получим:



Заметим, что символ «черный смайлик», как и многие другие, становится доступен при подключении в преамбуле соответствующего пакета символьных шрифтов (в нашем случае пакета `wasysym`). Для каждого символа из пакета определена своя команда, например наш смайлик будет вызываться командой `\blacksmiley`.

2.3. Команды для подключения семейства шрифтов

Чтобы изменить используемые в документе по умолчанию латинские шрифты Computer Modern достаточно в преамбуле подключить соответствующий пакет. Например, если требуется оформить статью в гарнитуре

Times, достаточно добавить в преамбулу команду `\usepackage{times}`. После этого все шрифты заменятся на шрифты типа таймс. Эта команда не меняет шрифтов, используемых в формулах, и они выглядят не очень хорошо (гарнитура таймс в среднем жирнее обычной Computer Modern). Чтобы изменить и шрифты для формул, в преамбуле дается команда `\usepackage{mathptmx}`.

Этот способ имеет два недостатка: соответствующие пакеты часто меняются и возможности легко использовать различные гарнитуры доступны лишь при использовании латинских шрифтов.

Подключать к LaTeX'у набор новых шрифтов таким образом, чтобы в `tex`-файле можно было менять их начертание и размер, можно при помощи приведенных ниже команд.

Семейство шрифтов подключается с помощью команды `\fontfamily{xxx}`, где `xxx` — внутренне имя шрифта (например, вместо `xxx` указывается `ptm` — внутреннее имя семейства шрифтов типа таймс). Команда `\selectfont` переключает шрифт. При этом остальные параметры (размер, насыщенность, начертание) остаются неизменными.

Если требуется изменить эти параметры, то желаемые параметры шрифта устанавливаются до `\selectfont` при помощи команд `\fontseries{...}`, `\fontshape{...}` и `\fontsize{...}{...}`.

Начертание изменяется с помощью команды `\fontshape`: курсивное начертание (аргумент `it`), прямое начертание (аргумент `n`).

Насыщенность меняется с помощью команды `\fontseries`. При этом аргумент `b` означает полужирный шрифт, `bx` – еще более насыщенный, обычный шрифт обозначается `m`.

Команда `\fontsize{...}{...}` явно указывает кегль и интерлиньяж шрифта. Например, команда `\fontsize{14pt}{24pt}` позволяет напечатать две строки шрифтом с размером 14pt и расстоянием между строками 24pt.

Команды `\fontfamily`, `\fontseries` и `\fontshape` и `\fontsize` являются командами «нижнего уровня», поэтому, по возможности следует использовать не их, а команды типа `\textbf`, `\textit`, `\textsf` или

`\rmfamily`, `\sffamily`, `\upshape`, `\bfseries`.

И, наконец, если требуется, чтобы основным шрифтом документа был определенный шрифт с внутренним именем **xxx**, например, шрифт без засечек `cmss`, то можно переопределить в преамбуле команду `\rmdefault`. По умолчанию команда `\rmdefault` задает шрифт Computer Modern с внутренним именем `cmr`, а переопределяется она в нашем случае так: `\renewcommand*{\rmdefault}{cmss}`.

3. Коллекции кириллических шрифтов (Cyrillic fonts) в LaTeX

Ниже мы рассмотрим несколько наиболее популярных коллекций кириллических шрифтов.

3.1. Кириллические шрифты LH

Лучшими кириллическими шрифтами CyrTeXовская общественность считает шрифты LH.

Их авторы Ольга Лапко (Издательство Мир) и Андрей Ходулев разработали бесплатный шрифт, который позволяет оформлять научные тексты с помощью LaTeXсовершенно не заботясь о подборке согласованных шрифтов для формул и текста. Основные гарнитуры этих шрифтов входят в последние дистрибутивы LaTeX. Полный комплект шрифтов LH всегда можно достать по адресу `CTAN:fonts/cyrillic/lh`.

Последняя версия LH поддерживает стандартные кодировки LaTeXT2A, T2B, T2C, X2 (кроме перечисленных поддерживаются LCY, OT2, и др.).

Для того, чтобы использовать стандартные гарнитуры LH, достаточно просто подключить русский язык в преамбуле документа. Чтобы начать работу с полной коллекцией шрифтов LH, необходимо в преамбуле TeX'овского документа установить внутреннюю кодировку символов, например, T2A, дав команду `\usepackage[T2A]{fontenc}`, и подключить соответствующий стилевой файл `\usepackage{lh-t2accr}`.

При этом по умолчанию будут подключены: шрифт с засечками — обычный Roman, рубленый шрифт — Sans serif, шрифт пишущей машин-

ки — Typewriter. Для переключения большинства начертаний используются стандартные команды LaTeX2e для управления насыщенностью, размером и начертанием шрифтов:

```
\rmfamily
\mdseries\upshape {Обычный} (roman),
\slshape наклонный (slanted),
\itshape курсив (italic),
\scshape капитель (small caps),
\fontseries b\upshape полужирный «суженный» (bold),
\bfseries\upshape полужирный (bold extended),
\itshape полужирный курсив (bold italic),
\slshape полужирный наклонный (bold slanted),
\scshape полужирная капитель (bold small caps),
\mdseries\fontshape{ui}\selectfont
«прямой» курсив (upright italic).

\sffamily
\mdseries\upshape Рубленый (sans serif),
\slshape наклонный (slanted),
\bfseries\upshape полужирный,
\slshape полужирный наклонный (bold slanted),
\fontseries{sbc}\upshape
полужирный «суженный» (semibold condenced).

\ttfamily
\mdseries Пишущая машинка (typewriter)
\slshape наклонный (slanted),
\itshape курсив (italic),
\scshape капитель(small caps).
```

Результат применения этих команд выглядит следующим образом:

- Обычный (roman), *наклонный (slanted), курсив (italic),* КАПИТЕЛЬ (SMALL CAPS), **полужирный «суженный» (bold), полужирный (bold extended), *полужирный курсив (bold italic), полужирный наклонный (bold slanted),*** ПОЛУЖИРНАЯ КАПИТЕЛЬ (BOLD SMALL CAPS), «прямой» курсив (upright italic).
- Рубленый (sans serif), *наклонный (slanted),* **полужирный, *полужирный наклонный (bold slanted),* полужирный «суженный» (semibold condenced).**
- Пишущая машинка (typewriter) *наклонный (slanted), курсив (italic),* КАПИТЕЛЬ(SMALL CAPS).

Помимо перечисленных, коллекция LH включает в себя еще несколько гарнитур русских шрифтов. Для того, чтобы LaTeXначал их использовать, нужно явно указать ему внутреннее имя шрифта. Это можно сделать используя команду `\fontfamily{xxx}`, где `xxx` — (внутреннее имя шрифта, которое может состоять из 3 и более символов). Например:

`lcmss` Рубленый шрифт для слайдов (slides sans serif), прямой, наклонный (slanted), полужирный (bold);

`lcmtt` Шрифт пишущей машинки для слайдов (slides typewriter), прямой, курсив (italic), невидимый шрифт пишущей машинки;

`lcmss` Рубленый шрифт для слайдов (slides sans serif), прямой, наклонный (slanted), полужирный (bold), невидимый рубленый шрифт для слайдов (invisible ss), невидимый наклонный (invisible slanted);

`cmd` Данхилл (Danhill), одно начертание;

`cmfib` Фибоначчи (Fibonacci), прямой и наклонный (slanted);

Шрифт Фибоначчи, *увы, имеет лишь* прямое *и наклонное начертания*.

```
\fontfamily{cmfib}\mdseries\upshape
Шрифт Фибоначчи, \textsl{увы, имеет лишь
\emph{прямое}  и наклонное начертания}.
```

`cmfr` Забавный (funny roman), прямой и курсив (italic);

Забавный шрифт хорош для таких забавных текстов, как этот:

Варкалось, хливкие шорьки
Пырялись по наве,
И хрюкотали зелюки,
Как мюмзики в мове.

`cmvtt` Разноширинный шрифт пишущей машинки (variable typewriter), прямой и курсив (italic);

`cmbr` Шрифт семейства CM Bright, прямой, наклонный (slanted), полужирный (semibold, bold);

`cmtl` Шрифт семейства CM Bright Typewriter, прямой и наклонный (slanted);

`ccr` Шрифт для «Конкретной математики».

Кириллические шрифты пакета LH, предназначенные для издания научных текстов, продолжают развиваться — не так давно было добавлено новое начертание Concreate — Конкретная Математика:

```
\fontfamily{ccr}
\mdseries\upshape
Шрифт для «Конкретной математики»(Concrete roman):

\slshape наклонный (slanted), \itshape курсив (italic),
\scshape капитель (small caps), \bfseries\upshape
полужирный (bold), \itshape полужирный курсив (bold
italic), \slshape полужирный наклонный (bold slanted),
\scshape полужирная капитель (bold small caps),
\fontseries c\slshape суженный наклонный
(condensed slanted).
```

Результат применения приведенных выше команд выглядит следующим образом:

Шрифт для «Конкретной математики»(Concrete roman):

наклонный (slanted), курсив (italic), КАПИТЕЛЬ (SMALL CAPS), **полужирный (bold), *полужирный курсив (bold italic), полужирный наклонный (bold slanted),* полужирная капитель (bold small caps),** *суженный наклонный (condensed slanted).*

3.2. Пакет CM-Super — коллекция PostScript-шрифтов

Пакет CM-Super, автором которого является Владимир Волович, содержит свободно распространяемые и легальные PostScript-шрифты. Особенность этих шрифтов состоит в том, что они получены преобразованием свободно распространяемых METAFONT-шрифтов (включая LH-шрифты,

шрифты Computer Modern и другие). С точки зрения расширения ассортимента шрифтов этот пакет не особенно полезен, поскольку все входящие в него шрифты есть и в METAFONT-формате. Польза от него в другом: с его помощью можно готовить в системе TeX PostScript- и PDF-файлы, в которых использованы только шрифты Type 1. Такие файлы лучше приспособлены к просмотру в программе Acrobat Reader и к печати на принтерах с высоким разрешением.

3.3. PSCyr — коллекция русифицированных PostScript-шрифтов в формате Type 1

В настоящее время получили распространение два пакета, позволяющие использовать векторные (Type 1) варианты базовых TEX'овских шрифтов (Computer Modern) для набора русских текстов: `cmcyr` и `wncyr`. Работа с пакетом PSCyr (версии 0.4d) позволяет получить доступ ещё к *пятнадцати* семействам шрифтовых гарнитур в формате Type 1.

Полное число представленных базовых начертаний — 42, которое с учетом производимых из них вариантных начертаний (наклонный шрифт и капитель) дает полное число доступных шрифтов, равное 71. Набор имеющихся в шрифтах глифов позволяет набирать тексты на русском, украинском и белорусском языках. Коллекция PSCyr включает в себя следующие гарнитуры.

Декоративные шрифты:

- **Кладезь (CooperPSCyr), одно начертание**
- **Рекламный (AdvertisementPSCyr), одно начертание**

```
\textcp{Кладезь (CooperPSCyr), одно начертание}

\textad{Рекламный (AdvertisementPSCyr), одно начертание}
```

Шрифты с засечками:

- Академическая (AcademyPSCyr); начертания: Прямое нормальное, **Полужирное**, *Курсивное*
- Антиква (AntiquaPSCyr); начертания: Прямое нормальное, **Полужирное**, *Курсивное*, ***Полужирное Курсивное***
- Балтика (HandbookPSCyr); начертания: Прямое нормальное, **Полужирное**, *Курсивное*
- Бодони (CollegePSCyr); начертания: Прямое нормальное, **Полужирное**, *Курсивное*

```
\textac{Академическая (AcademyPSCyr); начертания:
       Прямое нормальное, \textbf{Полужирное},
       \textit{Курсивное}}

\textaq{Антиква (AntiquaPSCyr); начертания:
     Прямое нормальное, \textbf{Полужирное},
     \textit{Курсивное},
     \textbf{\textit{Полужирное Курсивное}}}

\textha{Балтика (HandbookPSCyr); начертания:
Прямое нормальное, \textbf{Полужирное},
      \textit{Курсивное}}

\textco{Бодони (CollegePSCyr); начертания:
Прямое нормальное, \textbf{Полужирное},
       \textit{Курсивное}}
```

Шрифты с засечками:

- Журнальная (JournalPSCyr); начертания: Прямое нормальное, **Полужирное**, *Курсивное*
- Лазурская (Lazurski); начертание: Прямое нормальное
- Сувенир (SouvenirPSCyr); начертания: Прямое нормальное, **Полужирное**
- Таймс (TimesNewRomanPSMT); начертания: Прямое нормальное, **Полужирное**, *Курсивное*, ***Полужирное Курсивное***

```
\textjn{Журнальная (JournalPSCyr); начертания:
      Прямое нормальное,
      \textbf{Полужирное},
      \textit{Курсивное}}

\textlz{Лазурская (Lazurski); начертание:
       Прямое нормальное}

\textsv{Сувенир (SouvenirPSCyr); начертания:
      Прямое нормальное,
      \textbf{Полужирное}}

\texttm{Таймс (TimesNewRomanPSMT); начертания:
      Прямое нормальное,
      \textbf{Полужирное},
      \textit{Курсивное},
      \textit{\textbf{Полужирное Курсивное}}}
```

Рубленые шрифты:

- Ариал (ArialMT); начертания:
  Прямое нормальное, **Полужирное**, *Курсивное*, ***Полужирное Курсивное***, **Жирное**
- Букварная (TextbookPSCyr); начертания: Прямое нормальное, **Полужирное**, *Курсивное*
- Журнальная Рубленая (MagazinePSCyr); начертания:
  Прямое нормальное, **Полужирное**, *Курсивное*

```
\textar{Ариал (ArialMT);
начертания:\\
       Прямое нормальное,
       \textbf{Полужирное},
       \textit{Курсивное},
       \textit{\textbf{Полужирное Курсивное}},
       {\fontseries{eb}\selectfont Жирное}}

\texttx{Букварная (TextbookPSCyr);
начертания:
       Прямое нормальное,
       \textbf{Полужирное},
       \textit{Курсивное}}

\textma{Журнальная Рубленая (MagazinePSCyr);
       начертания:\\
       Прямое нормальное,
       \textbf{Полужирное},
       \textsl{Курсивное}}
```

Моноширинные шрифты:

- Курьер (CourierNewPSMT); четыре начертания: Прямое нормальное, **Полужирное**, *Курсивное*, ***Полужирное Курсивное***
- ER Курьер (ERKurierPSCyr); четыре начертания: Прямое нормальное, **Полужирное**, *Курсивное*, ***Полужирное Курсивное***

```
{\crfamily Курьер (CourierNewPSMT); четыре
       начертания: Прямое нормальное,
       \textbf{Полужирное}, \textit{Курсивное},
       \textit{\textbf{Полужирное Курсивное}}}

\texter{ER Курьер (ERKurierPSCyr); четыре
       начертания: Прямое нормальное,
       \textbf{Полужирное}, \textit{Курсивное},
       \textit{\textbf{Полужирное Курсивное}}}
```

Чтобы начать работу с коллекцией шрифтов PSCyr, необходимо в преамбуле TEX'овского документа подключить стилевой файл `pscyr.sty` и установить внутреннюю кодировку символов T2A дав команду `\usepackage[T2A]{fontenc}`. При этом по умолчанию будут подключены следующие шрифты: шрифт с засечками — Антиква, рубленый шрифт — Букварная, шрифт пишущей машинки — ERKurier. Эти шрифты неплохо сочетаются друг с другом и с математическими шрифтами системы TEX.

Для переключения начертаний следует пользоваться стандартными командами LaTeX2e: (`\textbf{...}`, `\textit{...}`, `\textsl{...}`, `\textsc{...}`, `\textsf{...}`, `\texttt{...}`). При необходимости набрать часть текста другим шрифтом, соответствующий блок текста поме-

щается в аргумент команды переключения шрифта: например, блок текста внутри команды `\textac{...}` будет набран шрифтом Academy.

Если же вы хотите набирать весь текст другим шрифтом, то разумнее будет сразу же в преамбуле документа установить соответствующий шрифт по умолчанию.

Например, команда `\renewcommand{\rmdefault}{ftm}` установит в качестве шрифта с засечками шрифт Таймс. Внутренние имена всех доступных шрифтов коллекции указаны в файле `pscyr.sty`; они кодируются тремя буквами, первой из которых является буква f (что значит free), а две последующие буквы получаются из сокращенного названия шрифта.

3.4. URWCyr — коллекция кирилизированных шрифтов компании URW

Пакет URWCyr предназначен для работы с кирилизированными шрифтами компании URW в системе LaTeX. Автором шрифтов URWCyr является А. Маляренко (`anatoliy.malyarenko@mdh.se`), а шрифтов URW — В. Филиппов, (`frob@df.ru`). Из коллекции URWCyr доступны следующие гарнитуры:

А. Шрифты с засечками. В эту группу входят гарнитуры URW Bookman; Century Schoolbook; URW Palladio; Nimbus Roman №9. Их аббревиатуры, соответственно, `ubk`, `unc`, `upl`, `utm`.

Б. Шрифты без засечек. Названия шрифтов: URW Gothic и Nimbus Sans, аббревиатуры, соответственно, `uag` и `uhv`.

В. Имитация шрифтов пишущей машинки. Шрифт называется Nimbus Mono и имеет внутреннее имя (аббревиатуру) `ucr`.

Для использования шрифтов в документе нужно установить соответствующий шрифт в преамбуле документа по умолчанию (или в нужном месте документа) при помощи команд

```
\renewcommand{\rmdefault}{aaa}
\renewcommand{\sfdefault}{bbb}
\renewcommand{\ttdefault}{ucr}
```

В аргументы команд вместо `aaa` можно подставлять произвольную аббревиатуру из группы А, а вместо `bbb` — произвольную аббревиатуру из группы Б.

Хорошие результаты получаются при использовании следующих стандартных комбинаций:

```
\renewcommand{\rmdefault}{ubk}
\renewcommand{\sfdefault}{uag}
\renewcommand{\ttdefault}{ucr}

\renewcommand{\rmdefault}{unc}
\renewcommand{\sfdefault}{uag}
\renewcommand{\ttdefault}{ucr}

\renewcommand{\rmdefault}{upl}
\renewcommand{\sfdefault}{uhv}
\renewcommand{\ttdefault}{ucr}

\renewcommand{\rmdefault}{utm}
\renewcommand{\sfdefault}{uhv}
\renewcommand{\ttdefault}{ucr}
```

Следующие несколько примеров иллюстрируют возможность разных способов задания начертания и насыщенности шрифта для некоторых наиболее экзотических гарнитур.

Уменьшение размера шляпы. Ваша шляпа великовата? *Приклейте липкий пластырь на внутреннюю поверхность шляпы. Возможно, вам потребуется два-три слоя, в зависимости от разницы в размерах. Кроме того, в жаркие дни он будет впитывать пот со лба.*

Очень полезный совет.

```
\renewcommand{\rmdefault}{ucr}
\textsl{\bfseries Уменьшение размера шляпы. }
Ваша шляпа великовата?
{\slshape Приклейте липкий пластырь на внутреннюю
поверхность шляпы.  Возможно, вам потребуется
два-три слоя, в зависимости от разницы в размерах.
Кроме того, в жаркие дни он будет впитывать
пот со лба.}

\hfill {\scshape Очень полезный совет.}
```

Недостаток, умноженный на *недостаток*, дает **наличие**.
Недостаток же, умноженный на НАЛИЧИЕ, дает **недостаток**.

Диофант А.

```
\renewcommand{\rmdefault}{utm}
\textsl{Недостаток}, умноженный на \textsl{недостаток},
дает \textbf{наличие}. \\
Недостаток же, умноженный на
\textsc{наличие}, дает \textbf{недостаток}.

\hfill {\itshape Диофант А.}
```

Загадка

Долго я следил за ней.
А она с утра до ночи
то сама себя длинней,
ТО САМА СЕБЯ КОРОЧЕ.

```
\renewcommand{\rmdefault}{uag}
{\bfseries Загадка}

Долго я следил за ней.\\А она с утра до ночи \\
\textsl{то сама себя длинней},\\
\textsc{то сама себя короче}.
```

4. Как выбрать подходящий шрифт?

Выбор той или иной гарнитуры шрифта определяется самим изданием, его целью и читательским адресом. В научном издании уместны одни шрифты, в научно-популярном издании –– другие, в художественном — третьи. На выбор шрифта влияет возраст и социальный статус аудитории, большое значение имеют также правильный выбор длины строки, размеров пробелов между словами, кегля шрифта, способа печати (некоторые шрифты из-за очень тонких засечек плохо воспроизводятся офсетной печатью) и даже бумаги (при представлении продукции в бумажном виде).

Решающим требованием, которому должен удовлетворять шрифт является удобочитаемость текста. Удобочитаемость шрифта определяется скоростью (быстротой восприятия) и удобством чтения как отдельных знаков, так и текста в целом, а также правильностью понимания прочитанного без лишнего напряжения и повышенной утомляемости.

Различные исследования показали, что:

- в печатном тексте шрифты с засечками читаются легче, так как засечки помогают взгляду передвигаться от буквы к букве, и буквы при этом не сливаются друг с другом;
- буквы без засечек легче читать в шрифтах очень большого или очень малого размера;
- текст, набранный прописными буквами, замедляет скорость чтения, а кроме того, занимает больше места, порой на 50 процентов;
- курсив читать труднее, чем прямое начертание;

- очень короткие строки, так же как и очень длинные, читать труднее.
- черезмерное разнообразие шрифтов в тексте не облегчает, а усложняет чтение.

Некоторые шрифты настолько универсальны, что подходят практически для любой задачи. У других более ограниченное применение. В частности, Литературную, Банниковскую, Лазурскую гарнитуры используют при наборе чаще всего, потому что в их рисунке сохраняется гротесковая выразительность шрифтов без засечек, но они более удобочитаемые, следовательно, ими можно набирать большие объемы текста. Шрифты Литературной гарнитуры сохраняют хорошую удобочитаемость во всех кеглях.

Лекция 5 ⏮

Расширенные возможности набора математического текста

Тонкая настройка в математическом режиме: выравнивание и нумерация многострочных формул, установка счетчика формул, окружения типа теоремы. Рисование схем и диаграмм средствами LaTeX и $\mathcal{A}\mathcal{M}\mathcal{S}$-LaTeX.

Формулы в TeX'е делятся на два вида: формулы *внутри текста* и *выключные*, т. е. вынесенные в отдельную строку.

Внутритекстовые формулы окружаются с обеих сторон знаками $ или заключаются между `\begin{math}` и `\end{math}`. TeX автоматически сжимает внутритекстовые формулы так, чтобы они заняли как можно меньше места по высоте. Для математических символов, функций, операций и операторов определены специальные команды LaTeX'а. Например, текст:

```
Истинный математик~-- $\tau\epsilon\chi$-нолог
отдает свое $\heartsuit$ математической моде,
в которой легко написать, что
\begin{math}\sin^2 \alpha+\cos^2 \alpha=1\end{math}.
```

после компиляции будет иметь вид:

> Истинный математик – $\tau\epsilon\chi$-нолог отдает свое $\heartsuit$ математической моде, в которой легко написать, что $\sin^2 \alpha+\cos^2 \alpha=1$.

Выключные формулы окружаются знаками `$$` или парой команд `\[` и `\]` (или заключаются между `\begin{displaymath}` и `\end{displaymath}`). Использование команд `\[` и `\]` позволяет одной стилевой опцией во всём документе прижать выключные формулы влево, тогда как формулы, заключённые между `$$` и `$$`, всегда центрированы.

Формулы нельзя разрывать пустой строкой.

Выключные формулы TEX печатает с временным прерыванием текущего абзаца, поэтому когда после формулы нет пустой строки или команды `\par`, следующая за формулой строка печатается без абзацного отступа.

В стандартных классах документа по умолчанию формулы располагаются по центру строки. Включение опции `fleqn` в команде `\documentclass[fleqn]{article}` выравнивает все формулы по левому краю страницы. При этом расстояние от левого поля страницы до формулы задаётся параметром `\mathindent` (по умолчанию 2.5 em), меняя значение которого командами `\setlength` или `\addtolength`, можно устанавливать любую величину отступа формулы от края страницы.

1. Тонкости набора формул

1.1. Маленькие хитрости набора сложных конструкций

Математические символы

LATEX 2_ε и $\mathcal{AMS}$-LATEX содержат практически все наиболее часто употребляемые математические символы.

Дополнительный набор разнообразных символов можно найти в пакетах `txfonts` (гарнитура Times) и `pxfonts` (гарнитура Adobe Palatino). Они содержат как экзотические (типа ♡), так и достаточно распространенные символы. Указанные пакеты переопределяют все команды для набора символов из пакетов `latexsym`, `amssymb` и `amsfonts` (математические алфавиты). Поэтому после загрузки пакета `txfonts` можно использовать символы из `latexsym`, `amssymb` и `amsfonts` не загружая сами пакеты. В частности, пакет `txfonts` содержит команды `\varg`, `\varv`, `\varw` и `\vary` для печати латинских букв g, v, w и y с другой формой символов. Чтобы изменить начертание этих букв во всем тексте, достаточно указать опцию `varg` при подключении пакета: `\usepackage[varg]{txfonts}`.

Различные математические, физические и технические символы содержатся также в пакетах `stmaryrd` и `wasysym`. В `wasysym` можно найти астрономические и астрологические знаки.

Полный свод символов (2266 штук) вместе с командами для их набора, доступных как в текстовой, так и в математической моде, можно найти

в файлах `symbols.tex` и `symbols.pdf` на Comprehensive TEX Archive Network (http://www.ctan.org или его зеркала) в директории `tex-archive/info/symbols/comprehensive`.

Для этого есть команды!

Для набора кратных интегралов имеются специальные команды. Набор при помощи `\int\int\int\int}` дает некрасивые результаты $\int\int\int\int$. Действительно:

```
\[
  \int\int\int f(x)\,dx
  \text{  --- это некрасиво!!!}
\] \[\iint\limits_X f(x)\,dx   \quad
     \iiint\limits_X f(x)\,dx  \quad
     \iiiint\limits_X f(x)\,dx \quad
     \idotsint\limits_X f(x)\,dx
   \]
```

$$\int\int\int f(x)\,dx \text{ — это некрасиво!!!}$$

$$\iint\limits_X f(x)\,dx \quad \iiint\limits_X f(x)\,dx \quad \iiiint\limits_X f(x)\,dx \quad \idotsint\limits_X f(x)\,dx$$

Не следует путать отношение параллельности $\parallel$ и скобку вида $\|$. И не стоит пытаться использовать пару символов `||` для их получения.

```
$AB \parallel CD$\qquad $\|a_{ij}\|$,
но не $AB\| CD$\qquad $||a_{ij}||$
```

$$AB \parallel CD \qquad \|a_{ij}\|, \text{ но не } AB\|CD \qquad ||a_{ij}||$$

Следует отличать бинарную операцию «$:$ » и двоеточие:

```
$\{C\colon I^p_e \to I_f\}$,
но не $\{C : I^p_e \to I_f\}$!
```

$$\{C\colon I^p_e \to I_f\}, \text{ но не } \{C : I^p_e \to I_f\}!$$

Одно над другим

Команды `\overset` и `\underset` позволяют поместить символы меньшего размера над и под формулой, а команда `\sideset` — поместить символы меньшего размера со всех сторон большого оператора:

```
\[
\overset{*}{X} \qquad \underset{*}{X} \qquad
\underset{i=1}{\overset{\infty}{X_i}}\qquad
\sideset{_1^2}{_3^4} \prod_k \qquad
\sideset{}{'}\sum_{0\le i\le m}E_i\beta x
\]
```

$$\overset{*}{X} \qquad \underset{*}{X} \qquad \underset{i=1}{\overset{\infty}{X_i}}\qquad \sideset{_1^2}{_3^4} \prod_k \qquad \sideset{}{'}\sum_{0\le i\le m}E_i\beta x$$

Команда `\substack` создает «многоэтажные» индексы:

```
\[
\sum_{\substack{n_1, n_2, \dots, n_r \\
n_1+n_2+\dots+n_r=n\\n_1,n_2,\dots,n_r>0}}
\frac{n!}{n_1!\,n_2!\dots n_r!}
\]
```

$$\sum_{\substack{n_1, n_2, \dots, n_r \\ n_1+n_2+\dots+n_r=n\\n_1,n_2,\dots,n_r>0}} \frac{n!}{n_1!\,n_2!\dots n_r!}$$

Команды `\xrightarrow` и `\xleftarrow` позволяют создавать растя-

жимые стрелки с надписями:

```
$X \xrightarrow [B\in\mathfrak{M}^0,C\in\mathfrak{M}^1]
{A = C\circ B} Y$\qquad $Y \xleftarrow{A = C\circ B} X$
```

$$X \xrightarrow[B\in\mathfrak{M}^0,C\in\mathfrak{M}^1]{A=C\circ B} Y \qquad Y \xleftarrow{A=C\circ B} X$$

Команды `\underbrace` и `\overbrace` позволяют печатать фигурные скобки, соответственно, под и над формулой. Немного сложнее изобразить фигурные скобки и под, и над формулой:

```
$\underbrace{\overbrace{0 1 2 \dots 9}^{10}
\rm A B \dots F}_{16}$
```

$$\underbrace{\overbrace{012\dots9}^{10}\mathrm{AB}\dots\mathrm{F}}_{16}$$

Перекрытие фигурных скобок требует преодоления TEX-нических трудностей, поскольку группы не могут перекрываться. Команда `\phantom` производит невидимую формулу, над которой рисуется скобка. Команда `\lefteqn` декларирует, что эта формула не занимает места по ширине:

```
$\lefteqn{\underbrace{\phantom{a_1+\dots+a_i+a_j}}}
a_1+\dots+\overbrace{a_i+a_j+\dots+a_n}$
```

$$\underbrace{a_1+\cdots+\overbrace{a_i+a_j}}+\cdots+a_n$$

Существует общая команда для дробей, биномиальных коэффициентов и аналогичных конструкций — команда `\genfrak`.

$$\left[\frac{1+x}{1-x}\right] = \left(\tfrac{1+x}{1-x}\right) = \left\langle\tfrac{1+x}{1-x}\right\rangle = \tfrac{1+x}{1-x}$$

Команда `\genfrak` имеет шесть аргументов: левая скобка, правая скобка, толщина дробной черты, номер стиля от 0 до 4, числитель, знаменатель. Для нашего примера аргументы будут такими:

```
\[
\genfrac{[}{]}{1pt}{0}{1+x}{1-x} =
\genfrac{(}{)}{}{1}{1+x}{1-x} =
\genfrac{<}{>}{}{2}{1+x}{1-x} =
\genfrac{}{}{}{3}{1+x}{1-x}
\]
```

Большое, среднее, маленькое

В математической моде используются символы трех размеров: текстового, индексов и индексов к индексам. Если размер шрифта документа равен 10 pt, то размер индексов и индексов к индексам будет, соответственно, 7 pt и 5 pt. Размер символов, которыми печатается та или иная часть формулы, задаётся её стилем.

TEX сам выбирает, какой стиль использовать в числителе и знаменателе дробей, в индексах и т.д., как например, при печати этой формулы:

```
\[
\sum_{i = 1}^{ \left[ \frac{n}{2} \right] }
\binom{ x_{i, i + 1}^{i^{2}} }
{ \left[ \frac{i + 3}{3} \right] }
\frac{ \sqrt{ \mu(i)^{ \frac{3}{2}} (i^{2} - 1) } }
{ \sqrt[3]{\rho(i) - 2} + \sqrt[3]{\rho(i) - 1} }
\]
```

$$\sum_{i=1}^{\left[\frac{n}{2}\right]} \binom{x_{i,i+1}^{i^2}}{\left[\frac{i+3}{3}\right]} \frac{\sqrt{\mu(i)^{\frac{3}{2}}(i^2-1)}}{\sqrt[3]{\rho(i)-2}+\sqrt[3]{\rho(i)-1}}$$

Стиль формулы можно изменить вручную следующими декларациями:

- `\displaystyle` (выключной) задаёт стиль для формул на отдельной строке (выключной стиль). В этом стиле используется текстовый размер символов.
- `\textstyle` (текстовый) задаёт стиль для формул в тексте и в окружение array. В этом стиле также используется текстовый размер символов.
- `\scriptstyle` (индексный) задаёт стиль для индексов первого уровня. В этом стиле используется размер индексов.
- `\scriptscriptstyle` (подиндексный) задаёт стиль для индексов высших уровней. В этом стиле используется размер индексов к индексам.

Размер разделителей, выбранный TEX'ом с помощью команд `\left` и `\right`, часто бывает неудовлетворительным, к примеру завышенным, поскольку подгоняется под полную высоту формулы, включая индексы. В таких случаях лучше вручную задавать размеры разделителей, используя команды для левых разделителей: `\bigl, \Bigl, \biggl и \Biggl`, команды для правых разделителей: `\bigr, \Bigr, \biggr и \Biggr`, команды для разделителей, стоящих посередине, когда нужно вставить небольшой пробел по обе стороны от разделительного символа: `\bigm, \Bigm, \biggm и \Biggm`, и команды `\big, \Big, \bigg и \Bigg`.

Полученные при помощи приведенных выше команд вложенные скобки различного размера и формы

$$\Bigg(\bigg[\Big\{\bigg\|\Big\langle\heartsuit\Big\rangle\bigg\|\Big\}\bigg]\Bigg)$$

целесообразно использовать, например, в следующих случаях:

```
\[ \bigl| |x|-|y| \bigr|,\qquad
\Bigl\{C \Bigm| \bigcup_{p=0}^4 I_e^p\to I_f
\Bigr\},                    \qquad
\frac{x+1}x \bigg/ \frac{y}{y+1} \]
```

$$\big||x|-|y|\big|, \qquad \Big\{C \,\Big|\, \bigcup_{p=0}^{4} I_e^p \to I_f\Big\}, \qquad \frac{x+1}{x}\Big/\frac{y}{y+1}$$

Следующая формула набрана по всем правилам:

$$\sin z = z\lim_{p\rightarrow\infty}\prod_{k=1}^{p-1}\left[1-\frac{z^2}{4p^2\tan^2\displaystyle\frac{k\pi}{2p}}\right]$$

В этом можно убедиться, посмотрев на исходный текст:

```
\[ \sin z = z\lim_{p\rightarrow\infty}\prod_{k=1}^{p-1}
\left[1-\frac{z^2}{4p^2\tan^2
\displaystyle\frac{k\pi}{2p}}\right] \]
```

Однако она имеет слишком высоко простирающиеся квадратные скобки, которые необходимо «вручную» сначала уменьшить по высоте, а затем опустить. Новый исходный текст для формулы:

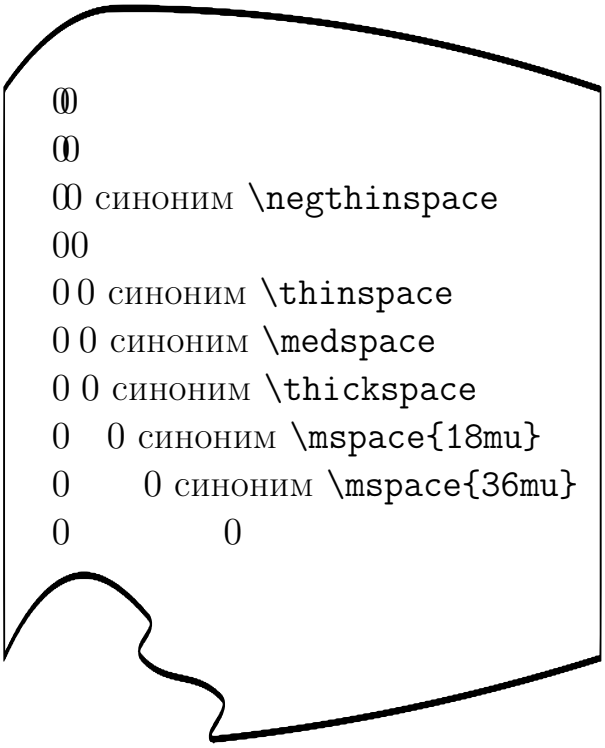
```
\[ \sin z = z\lim_{p\rightarrow\infty}\prod_{k=1}^{p-1}
\raisebox{-6pt}{$\left [\rule{0pt}{30pt}\right.$}
-\frac{z^2}{4p^2\tan^2\displaystyle\frac{k\pi}{2p}}
\raisebox{-6pt}{$\left ]\rule{0pt}{30pt}\right.$}
\]
```

$$\sin z = z\lim_{p\rightarrow\infty}\prod_{k=1}^{p-1}\left[-\frac{z^2}{4p^2\tan^2\displaystyle\frac{k\pi}{2p}}\right]$$

Здесь высота скобки в конструкции «разделитель без пары» задана линейкой `\rule` высотой 22pt, а опущена скобка на 5 pt командой `\raisebox`.

Если выбранные TEX пробелы внутри формул неудовлетворительны, их можно подстраивать с использованием команд управления пробелами. Способы явного задания промежутков в математических формулах:

```
$0\negthickspace 0$                          \\
$0\negmedspace 0$                            \\
$0\! 0$     синоним \verb|\negthinspace|\\
$0 0$                                        \\
$0\, 0$     синоним \verb|\thinspace|    \\
$0\: 0$     синоним \verb|\medspace|     \\
$0\; 0$     синоним \verb|\thickspace|   \\
$0\quad 0$  синоним \verb|\mspace{18mu}|\\
$0\qquad 0$ синоним \verb|\mspace{36mu}|\\
$0\mspace{72mu} 0$
```



Если математическое выражение в строке состоит не из одной формулы, а из нескольких, то пробелы между ними надо расставлять вручную. Для этого обычно используется команда `\qquad`. Отрицательный тонкий пробел (команда `\!`) и положительный пробел необходимы, например, в следующих случаях:

$$F_n = F_{n-1} + F_{n-2}, \qquad n \ge 2.$$

$$R_i{}^j{}_{kn}, \quad \text{лучше, чем} \quad R_i{}^j{}_{kn};$$

$$n/\ln n, \quad \text{лучше, чем} \quad n/\ln n;$$

$$x > 0 \quad \text{для всех } x \quad \text{лучше, чем} \quad x > 0\text{для всех } x$$

Способ набора:

```
\[ F_n=F_{n-1}+F_{n-2},\qquad n\ge 2. \]
\[ R_i{}^j{}_{\!kn},\quad \text{ лучше, чем }\quad
R_i{}^j{}_{kn};\]
\[n/\!\ln n, \quad\text{ лучше, чем }\quad
n/\ln n;\] \[ x>0 \quad \text{ для всех } x \quad
\text{ лучше, чем }\quad x>0 \text{для всех } x\]
```

1.2. Набор выключных формул без номера

Для всех способов набора выключных формул знаки пунктуации после выделенной формулы ставятся в конце самой формулы, иначе этот знак появится в начале новой строки.

Формулы, которые должны печататься без порядкового номера на отдельной строке, можно набрать тремя способами:

- заключить формулу между командами `\[` и `\]` (либо `$$` и `$$`);
- поместить формулу в окружение `displaymath`;
- использовать LaTeX'овские окружения со звездочкой типа `\begin{equation*} ... \end{equation*}` и подобные им окружения со звездочкой пакета `amsmath` (как это будет показано ниже).

Окружение LaTeX'а `\begin{equation*} ... \end{equation*}`, как и первые два способа печати выключной формулы без номера, располагает формулу на отдельной строке и центрирует ее, создавая нужные

вертикальные пробелы до и после формулы. Однако эти способы не позволяют разместить длинную формулу — TEX не сумеет перенести ее на отдельную строку (выдаст сообщение о переполнении `Overfull ...`).

Рассмотрим ситуации, когда требуется набрать формулу без номера, которая должна размещаться более, чем на одной строке.

Системы уравнений

Во всех рассматриваемых окружениях точки переноса строк задаются командой `\\`, а точки выравнивания формул из разных строк по вертикали задаются одним или несколькими амперсантами `&`.

Для набора систем уравнений в LATEX'е для ненумерованных формул предусмотрено окружение `eqnarray*`. Выражение в пределах одной строки должно состоять из трёх частей (возможно пустых), разделённых амперсантами `&`. Каждая часть помещается в свой столбец. В левом и правом столбцах формулы, соответственно, прижимаются к правому и левому краю, а в среднем — центрируются:

```
\begin{eqnarray*}   f(x)&=
&\cos x\\f'(x)&=&-\sin x\\\int_{0}^{x}f(y)dy&=&\sin x
\end{eqnarray*}
```

$$\begin{eqnarray*} f(x) &=& \cos x \\ f'(x) &=& -\sin x \\ \int_{0}^{x} f(y)dy &=& \sin x \end{eqnarray*}$$

В пакете `amsmath` есть окружения `align*`, `alignat*` и `flalign*`, в которых не нужно обязательно разбивать уравнения на три части:

```
\begin{align*}   &f(x)=
\cos x\\f'(x)&=-\sin x\\\int_{0}^{x}f(y)dy=\sin x&
\end{align*}
```

$$f(x)=\cos x$$
$$f'(x)=-\sin x$$
$$\int_{0}^{x}f(y)dy=\sin x$$

В окружении `gather*` пакета `amsmath` все строки центрируются, а символ `&` не используется:

```
\begin{gather*}
f(x)=\cos x\\f'(x)=-\sin x\\\int_{0}^{x}f(y)dy=\sin x
\end{gather*}
```

$$\begin{gather*}
f(x)=\cos x\\f'(x)=-\sin x\\\int_{0}^{x}f(y)dy=\sin x
\end{gather*}$$

Окружение `split` пакета `amsmath` используется только внутри других окружений для набора выделенных формул, поскольку оно само не переключает TEX в математическую моду. Пример:

```
\begin{equation*}  \text{Так:}
\begin{split}  f(x)&=
\cos x\\f'(x)&=-\sin x\\\int_{0}^{x}f(y)dy&=\sin x
\end{split}
\quad\text{или так:}\quad
\begin{split}  &f(x)=
\cos x\\&f'(x)=-\sin x\\&\int_{0}^{x}f(y)dy=\sin x
\end{split}
\end{equation*}
```

$$\begin{equation*} \text{Так:}
\begin{split} f(x)&=
\cos x\\f'(x)&=-\sin x\\\int_{0}^{x}f(y)dy&=\sin x
\end{split}
\quad\text{или так:}\quad
\begin{split} &f(x)=
\cos x\\&f'(x)=-\sin x\\&\int_{0}^{x}f(y)dy=\sin x
\end{split}
\end{equation*}$$

Так:
$$\begin{aligned} f(x) &= \cos x \\ f'(x) &= -\sin x \\ \int_0^x f(y)dy &= \sin x \end{aligned}$$
или так:
$$\begin{aligned} f(x) &= \cos x \\ f'(x) &= -\sin x \\ \int_0^x f(y)dy &= \sin x \end{aligned}$$

Окружения `aligned`, `gathered` и `alignedat` пакета `amsmath`, как и `split`, используются только внутри других окружений для набора выделенных формул. В них формулы форматируются по тем же правилам, что и в окружениях `align`, `gather` и `alignat`, соответственно. В отличие от последних, они занимают по горизонтали не всю строку, а ровно столько, сколько необходимо для уравнений. Это позволяет размещать на одной строке несколько блоков уравнений, в каждом из которых формулы форматируются независимо друг от друга.

Расщепление длинных формул

Для того, чтобы разместить на несколько строк длинную выключную формулу (без номера), чаще всего используются: LaTeX'овское окружение `eqnarray*`; окружения, доступные при подключении пакета `amsmath`, а имено, `multline*`, `gather*`, `align*` (и подобные ему `alignat*` и `flalign*`); окружения пакета `amsmath`, не переключающие математическую моду: `split`, `aligned`, `gathered` и `alignedat`. Последние используются только внутри других окружений для набора выключных формул.

Длинные формулы, заключенные в эти окружения, не будут автоматически разбиваться на правильные части. необходимо явно указать, где их разбивать и насколько выравнивать.

В окружении `multline*` точки переноса строк задаются командой `\\`. Первая часть уравнения прижимается к левому полю, а последняя — к правому. Остальные части центрируются. Пример работы окружения `multline*`:

```
\begin{multline*}
A=a_{11}\xi_1^2+{}\\
{}+2a_{12}\xi_1\xi_2+3a_{13}\xi_1
\xi_3+4a_{14}\xi_1\xi_4+\dots+n a_{1n}\xi_1\xi_n+{}\\
{}+\sum_{i,j=2}^n a_{ij}\xi_i\xi_j
\end{multline*}
```

$$\begin{multline*}
A=a_{11}\xi_1^2+\\
+2a_{12}\xi_1\xi_2+3a_{13}\xi_1\xi_3+4a_{14}\xi_1\xi_4+\dots+na_{1n}\xi_1\xi_n+\\
+\sum_{i,j=2}^n a_{ij}\xi_i\xi_j
\end{multline*}$$

Расстояние от первой и последней строк до полей можно задавать, меняя значение параметра `\multlinegap`. Любую из средних строк можно сместить влево или вправо, сделав её аргументом команд `\shoveleft` и `\shoveright`, (символ `\\` остается вне).

В окружении `eqnarray*` точки переноса строк задаются командой `\\`, а точки выравнивания уравнений из разных строк по вертикали — символом `&`:

```
\begin{eqnarray*}                    A=a_{11}\xi_1^2+{}\\
&+2a_{12}\xi_1\xi_2+3a_{13}\xi_1\xi_3+
4a_{14}\xi_1\xi_4+5a_{15}\xi_1\xi_5+\dots +{}&n a_{1n}
\xi_1\xi_n+{}\\&&+\sum_{i,j=2}^n a_{ij}\xi_i\xi_j{}
\end{eqnarray*}
\begin{eqnarray*}                    A=a_{11}\xi_1^2+&{}&\\
+2a_{12}\xi_1\xi_2&+3a_{13}\xi_1\xi_3+
4a_{14}\xi_1\xi_4+5a_{15}\xi_1\xi_5+&\dots +{} n a_{1n}
\xi_1\xi_n+{}\\&{}&+{}\sum_{i,j=2}^n a_{ij}\xi_i\xi_j
\end{eqnarray*}
```

$$
\begin{aligned}
A = a_{11}\xi_1^2 &+ \\
&+2a_{12}\xi_1\xi_2 + 3a_{13}\xi_1\xi_3 + 4a_{14}\xi_1\xi_4 + 5a_{15}\xi_1\xi_5 + \cdots + \quad na_{1n}\xi_1\xi_n + \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad + \sum_{i,j=2}^{n} a_{ij}\xi_i\xi_j
\end{aligned}
$$

$$
\begin{aligned}
A = a_{11}\xi_1^2 &+ \\
+2a_{12}\xi_1\xi_2 \quad &+3a_{13}\xi_1\xi_3 + 4a_{14}\xi_1\xi_4 + 5a_{15}\xi_1\xi_5 + \quad \cdots + na_{1n}\xi_1\xi_n + \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad + \sum_{i,j=2}^{n} a_{ij}\xi_i\xi_j
\end{aligned}
$$

Заметим, что в последней формуле по обеим сторонам средней колонки (слева и справа от знаков $+$) получилось слишком много свободного места. Оно может быть уменьшено установкой:

```
{\setlength\arraycolsep{2pt}
\begin{eqnarray*}
A=a_{11}\xi_1^2+&{}&\\
+2a_{12}\xi_1\xi_2&+3a_{13}\xi_1\xi_3+
4a_{14}\xi_1\xi_4+5a_{15}\xi_1\xi_5+&\dots +{} n a_{1n}
\xi_1\xi_n+{}\\
&{}&+{}\sum_{i,j=2}^n a_{ij}\xi_i\xi_j
\end{eqnarray*}}
```

$$
\begin{aligned}
A = a_{11}\xi_1^2 &+ \\
+2a_{12}\xi_1\xi_2 \; &+3a_{13}\xi_1\xi_3 + 4a_{14}\xi_1\xi_4 + 5a_{15}\xi_1\xi_5 + \cdots + na_{1n}\xi_1\xi_n + \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad + \sum_{i,j=2}^{n} a_{ij}\xi_i\xi_j
\end{aligned}
$$

Чтобы части формулы на разных строках не начинались точно одна под другой, приходится использовать разные ухищрения, в частности, команды, вставляющие пробелы (типа `\qquad`).

В окружении `gather*` точки переноса строк также задаются командой `\\`. Все строки центрируются, а символ `&` не употребляется:

```
\begin{gather*}
A=a_{11}\xi_1^2+\\
+2a_{12}\xi_1\xi_2+3a_{13}\xi_1\xi_3+4a_{14}
\xi_1\xi_4+\dots +n a_{1n}\xi_1\xi_n+\\
+\sum_{i,j=2}^n a_{ij}\xi_i\xi_j
\end{gather*}
```

$$
\begin{gathered}
A=a_{11}\xi_1^2+\\
+2a_{12}\xi_1\xi_2+3a_{13}\xi_1\xi_3+4a_{14}\xi_1\xi_4+\dots +n a_{1n}\xi_1\xi_n+\\
+\sum_{i,j=2}^n a_{ij}\xi_i\xi_j
\end{gathered}
$$

Выбор альтернатив

Выбор из нескольких альтернатив можно задать при помощи окружения `array` и окружения `cases`, например так, как показано ниже:

```
\begin{displaymath}
\theta(x) =
\left\{ \begin{array}{ll}
a & \textrm{если $d>c$}\\
b+x & \textrm{по утрам}\\
l & \textrm{остальное время дня}
\end{array}
\right.
\end{displaymath}
```

```
\[  \theta(x) =
\begin{cases}
a & \textrm{если $d>c$}\\
b+x & \textrm{по утрам}\\
l & \textrm{остальное время дня}
\end{cases}
\]
```

После компиляции оба эти способа дадут один и то же результат:

$$\theta(x) = \begin{cases} a & \textrm{если $d>c$} \\ b+x & \textrm{по утрам} \\ l & \textrm{остальное время дня} \end{cases}$$

1.3. Автоматическая нумерация выключных формул

В математических текстах формулы обычно нумеруются для того, чтобы на них можно было ссылаться по ходу документа. LaTeX позволяет организовать нумерацию таким образом, чтобы номера формул и ссылки на них создавались автоматически.

Окружение `equation`

Чтобы LaTeX пронумеровал формулу, её надо поместить в окружение `equation` (без звездочки, которая всегда подавляет нумерацию). Каждая такая формула на печати автоматически получает свой порядковый номер:

$$\sin^2\varphi + \cos^2\varphi = 1; \tag{5.1}$$

$$F(x)|_a^b = F(b) - F(a); \tag{5.2}$$

$$\sum_{i=1}^{\left[\frac{n}{2}\right]} \binom{x_{i,i+1}^{i^2}}{\left[\frac{i+3}{3}\right]} \frac{\sqrt{\mu(i)^{\frac{3}{2}}(i^2-1)}}{\sqrt[3]{\rho(i)-2}+\sqrt[3]{\rho(i)-1}}. \tag{5.3}$$

Окружение multline

В окружении `multline` пакета `amsmath` формуле присваивается один номер, который печатается на последней строке:

$$A = a_{11}\xi_1^2 + \\ + 2a_{12}\xi_1\xi_2 + 3a_{13}\xi_1\xi_3 + 4a_{14}\xi_1\xi_4 + 5a_{15}\xi_1\xi_5 + \dots + na_{1n}\xi_1\xi_n + \\ + \sum_{i,j=2}^{n} a_{ij}\xi_i\xi_j \quad (5.4)$$

Окружения eqnarray, gather и другие

В LaTeX'овском окружении `eqnarray` каждая строка получает при печати свой номер.

$$A = a_{11}\xi_1^2 + \quad (5.5)$$

$$+2a_{12}\xi_1\xi_2 + \dots + na_{1n}\xi_1\xi_n + \quad (5.6)$$

$$+ \sum_{i,j=2}^{n} a_{ij}\xi_i\xi_j \quad (5.7)$$

Подавить нумерацию любой отдельной строки можно командой `\nonumber`. Например,

```
{\setlength\arraycolsep{2pt}
\begin{eqnarray}
A= a_{11}\xi_1^2+&{}&\nonumber\\
+2a_{12}\xi_1\xi_2&+\dots&+{}
n a_{1n}\xi_1\xi_n+{} \nonumber\\
&{}&+{}\sum_{i,j=2}^n a_{ij}\xi_i\xi_j
\end{eqnarray}}
```

после компиляции дает:

$$A = a_{11}\xi_1^2 + \\ +2a_{12}\xi_1\xi_2 + \dots + na_{1n}\xi_1\xi_n + \\ + \sum_{i,j=2}^{n} a_{ij}\xi_i\xi_j \quad (5.8)$$

Окружения `equation` и `eqnarray` используют один и тот же счетчик `equation`, значение которого и печатается как номер формулы. Следовательно, все нумерованные уравнения в документе будут иметь единую нумерацию.

Окружения пакета `amsmath`: `gather`, `align`, `alignat` и `flalign` создают нумерацию полностью аналогично.

Размещение номера формулы и ссылки на нее

В стандартных классах документа номер формулы по умолчанию ставится справа от формулы. Однако, при наличие опции `leqno` в команде `documentclass[leqno]{article}`, номера формул будут размещаться слева от формулы.

Пакет `amsmath` вводит опцию `reqno`: если какой-нибудь нестандартный класс устанавливает по умолчанию нумерацию с левой стороны уравнений, то опция `reqno` отменяет это правило. То, как именно выглядит на печати номер формулы, зависит от класса документа: например, в классе `article` формулы имеют сплошную нумерацию, а в классах типа `book` нумерация формул начинается заново в каждой главе и номер `n`-й по счёту пронумерованной формулы из главы `k`, имеет вид (`k.n`).

Номера уравнений хранятся в счётчике `equation`. С помощью команд, предназначенных для работы со счётчиками, можно самостоятельно менять формат номера формул и последовательность нумерации.

Один из вариантов изменения нумерации возможен при подключении пакета `amsmath`. Если в преамбулу документа вписать `\numberwithin{equation}{section}`, то в каждой секции уравнения будут нумероваться независимо друг от друга, и, например, номер формулы классе `article`, будет иметь вид (`k.n`).

Вместо `section` можно указать другие имена счётчиков команд секционирования. Для `subsection` номер формулы будет иметь вид (k.n.m).

Чтобы на присвоенный формуле номер можно было ссылаться в тексте, надо пометить ее, написав в любом месте `\begin{окружение}` и

`\end{окружение}` команду `\label{name}`, аргумент которой «name» (произвольная комбинация латинских символов) и будет именем формулы. Команда `\ref{name}` будет печатать номер этой формулы (без круглых скобок).

В пакете `amsmath` имеется команда `\eqref{name}`, которая сама ставит скобки. Например:

```
\begin{equation}\label{ogo1}
\theta(x) =
\begin{cases}
a & \textrm{если $d>c$}\\
b+x & \textrm{по утрам}\\
l & \textrm{остальное время дня}
\end{cases}
\end{equation}
Формула (\ref{ogo1}) некорректна, поскольку в формуле
\eqref{ogo1} не определено значение
$\theta(x)$ для полуночи.
```

$$\theta(x) = \begin{cases} a & \text{если } d > c \\ b + x & \text{по утрам} \\ l & \text{остальное время дня} \end{cases} \tag{1}$$

Формула (1) некорректна, поскольку в формуле (1) не определено значение $\theta(x)$ для полуночи.

Заметим в заключение, что нумеруемые формулы обязательно выделяются в отдельную строку. В отличие от LaTeX 2_ε, $\mathcal{AMS}$-LaTeX автоматически нумерует по порядку не все формулы без разбора, а только те, на которые есть ссылка в тексте; при добавлении или удалении ссылки нумерация меняется. В редких случаях возникает нужда обойти эту автоматику и средства для этого есть, но не следует ими увлекаться.

2. Матрицы

2.1. Возможности окружения array

Чтобы набрать матрицу, можно воспользоваться окружением `array`, поместив его в математическую моду (заметим, что само окружение не переключает TEX в математическую моду).

Отметим, что в окружении `array` набор формулы начинается в стиле `textstyle`, поэтому размер символов в числителе и знаменателе дроби сразу печатается в стиле индексов, как у внутритекстовой формулы.

Всякая матрица состоит из строк и столбцов; в исходном тексте строки матрицы разделяются командой `\\`, при этом последнюю строку заканчивать командой `\\` не нужно. Элементы внутри одной строки, относящиеся к разным столбцам, отделяются друг от друга символами `&`.

Сколько и каких столбцов должно быть в матрице, задаётся в обязательном аргументе команды `\begin{array}{вид и количество столбцов}`. Например, в этом случае аргумент представляет собой четыре буквы `cccc`:

```
\[ \begin{array}{cccc}
a_{11}& a_{12} &\ldots & a_{1n}\\
a_{21}& a_{22} &\ldots & a_{2n}\\
\vdots& \vdots &\ddots & \vdots\\
a_{n1}& a_{n2} &\ldots & a_{nn}
\end{array} \]
```

$$\begin{array}{cccc}
a_{11}& a_{12} &\ldots & a_{1n}\\
a_{21}& a_{22} &\ldots & a_{2n}\\
\vdots& \vdots &\ddots & \vdots\\
a_{n1}& a_{n2} &\ldots & a_{nn}
\end{array}$$

Количество букв соответствует количеству столбцов в матрице, а содержимое каждого из этих столбцов расположено по центру столбца (поскольку в аргументе только буквы `c`).

Кроме `c`, в аргументе окружения могут стоять буквы `l` или `r`, означающие, что содержимое этих столбцов будет выравнено по левому краю или по правому краю соответственно.

Скобки или иные разделители вокруг матрицы ставятся с помощью конструкции `\left` и `\right`. Например, чтобы заключить матрицу в круглые скобки, надо перед `\begin{array}` написать `\left (`, а после `\end{array}`, соответственно, `\right )`.

Окружение `array` перед обязательным аргументом может иметь необязательный аргумент с допустимыми значениями `t`, `c` (действует по умолчанию) и `b`. Он задает положение во вертикали осевой линии «матрицы». По умолчанию она проходит через середину. При `[t]` осевая линия самой матрицы совпадает с осевой линией её верхней строки, а при `[b]` — осевая линия самой матрицы совпадает с осевой линией её нижней строки. Сама матрица позиционируется в уравнении так, чтобы её осевая линия совпала с осевой линией уравнения (на уровне знака минус). Пример:

```
\[ A =
-\begin{array}[t]{c} a\\b \end{array},
-\begin{array}{c} a\\b \end{array},
-\begin{array}[b]{c} a\\b \end{array} \]
```

$$A = -\begin{array}{c} a \\ b \end{array}, -\begin{array}{c} a \\ b \end{array}, -\begin{array}[b]{c} a \\ b \end{array}$$

Промежутки между элементами матрицы можно регулировать. Параметр `\arraycolsep` равен половине величины горизонтального пробела между двумя колонками (по умолчанию равен 5 pt). Значением команды `\arraystretch` является вещественное число, равное по умолчанию 1. Величина вертикального пробела между рядами получается путем умножения этого числа на величину пробела, принятую по умолчанию. Если декларацией `\renewcommand` заменить значение `\arraystretch` на 1.25, то расстояние между строками на печати будет больше в 1.25 раза.

Помимо `l`, `c` и `r`, в аргументе окружения `array` можно использовать

символ `|`, который проводит вертикальную линию на всю высоту матрицы. Перед первой строкой, а также непосредственно вслед за `\\` и последней строкой можно поставить команду `\hline`, которая проведет горизонтальную линию на всю ширину матрицы. Горизонтальную линию через часть столбцов можно провести командой `\cline`. Вертикальную линию через одну строку можно провести, заменяя ячейку командой `\multicolumn` и используя в её аргументе символ `|`. Пример:

```
\[  C = \left( \begin{array}{ccc}1&0&0\\\cline{2-3}0&
\multicolumn{1}{|c}{0}&i\\0&\multicolumn{1}{|c}{-i}&0
\end{array}                 \right)          \]
```

$$C = \left(\begin{array}{ccc}1&0&0\\\cline{2-3}0&\multicolumn{1}{|c}{0}&i\\0&\multicolumn{1}{|c}{-i}&0\end{array}\right)$$

Толщину линий можно выбрать, изменяя значение параметра `\arrayrulewidth`.

2.2. Возможности пакета amsmath

Пакет `amsmath` вводит несколько специализированных окружений для набора матриц. В отличие от окружения `array` LaTeX'а, для них не надо указывать количество столбцов в матрице и задавать расположение формул в ячейках. Внутри столбцов они всегда центрируются. Элементы матрицы, как и в случае окружения `array`, разделяются символом `&`, строки разделяются командой `\\`.

В пакете `amsmath` базовая форма матрицы определена без скобок и задается с помощью окружения `matrix`:

```
\[\begin{matrix}
  a+b+c&uv&x-y&27\\a+b&u+v&z&134
  \end{matrix}    \]
```

$$\begin{matrix} a+b+c & uv & x-y & 27 \\ a+b & u+v & z & 134 \end{matrix}$$

Чтобы изобразить скобки, используются окружения `pmatrix`, `bmatrix`, `vmatrix`, и `Vmatrix`, которые автоматически окружают матрицу скобками: круглыми, квадратными, прямыми и двойными соответственно. Матрицы с различными типами окружающих скобок:

```
\[ \begin{matrix}  0 & 1 \\ 1 & 0 \end{matrix} \qquad
   \begin{pmatrix} 0 &-i \\ i & 0 \end{pmatrix}\qquad
   \begin{bmatrix} 1 & 0 \\ 0 &-1 \end{bmatrix}\qquad
   \begin{vmatrix} a & b \\ c & d \end{vmatrix}\qquad
   \begin{Vmatrix} 1 & 0 \\ 0 & 1 \end{Vmatrix}\]
```

$$\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix} \qquad \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \begin{vmatrix} a & b \\ c & d \end{vmatrix} \qquad \begin{Vmatrix} 1 & 0 \\ 0 & 1 \end{Vmatrix}$$

Перечисленные окружения при наборе формул могут сами входить в состав более сложных конструкций, наподобие следующей:

```
\[\mathbf{A}=
  \begin{pmatrix}
  a+b+c&uv\\ \begin{pmatrix}a&b\\c&d\end{pmatrix}&u+v
  \end{pmatrix}
  \begin{pmatrix}30&7\\3&17\end{pmatrix} \]
```

$$\mathbf{A}=\begin{pmatrix} a+b+c & uv \\ \begin{pmatrix} a & b \\ c & d \end{pmatrix} & u+v \end{pmatrix} \begin{pmatrix} 30 & 7 \\ 3 & 17 \end{pmatrix}$$

Для изображения матриц с многоточиями используются команды `\dots`, `\ddots`, `\vdots` и `\hdotsfor`. Команда `\hdotsfor` служит для печати строки точек в больших матрицах (называемой отточием). Обяза-

тельный аргумент этой команды задаёт число столбцов, занятых точками, а стоящий первым необязательный декларирует расстояние между точками в относительных единицах к стандартному. Простейшие виды матриц с многоточиями:

```
\[\begin{pmatrix}
 a_{11}&\dots&a_{1n}\\ \vdots&\ddots&\vdots\\
 a_{m1}&\dots&a_{mn}
 \end{pmatrix} \qquad\qquad \begin{pmatrix}
 a_{11}&\dots&a_{1n}\\\hdotsfor{3}\\a_{m1}&\dots&a_{mn}
\end{pmatrix}      \]
```

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \qquad\qquad \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}$$

Можно изобразить немного более сложную матрицу, используя окружение `Vmatrix` и отточие:

```
\[ W(\Phi) =
  \begin{Vmatrix}
  1&0&\hdotsfor{2}&0\\k_{n2}&1&0&\dots&0\\
  \hdotsfor[2]{5}\\k_{n1}&k_{n2}&\dots&k_{n\,n-1}&1
  \end{Vmatrix}     \]
```

$$W(\Phi) = \begin{Vmatrix} 1 & 0 & \dots & \dots & 0 \\ k_{n2} & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ k_{n1} & k_{n2} & \dots & k_{n\,n-1} & 1 \end{Vmatrix}$$

Команда `\dfrac` является вариантом `\frac`, но дает изображение в размере `displayed`.

Для создания этой формулы использованы пакеты `amsmath` и `amssymb`.

Матрица с метками строк и столбцов набирается при помощи команды **bordermatrix**. Здесь разделителем строк служит команда **\cr**, а сама матрица заключается в круглые скобки, как в следующем примере:

```
\[\bordermatrix{ &&&j&\cr
  &b_{11}&\dots&0&b_{1n}\cr &\dots&\dots&0&\dots\cr
  i&0&0&1&0\cr &\dots&\dots&0&\dots\cr
  &b_{m1}&\dots&0&b_{mn}\cr }\]
```

$$\bordermatrix{ &&&j&\cr
  &b_{11}&\dots&0&b_{1n}\cr &\dots&\dots&0&\dots\cr
  i&0&0&1&0\cr &\dots&\dots&0&\dots\cr
  &b_{m1}&\dots&0&b_{mn}\cr }$$

По умолчанию в матрицах не может быть больше 10 столбцов. Максимальное допустимое число столбцов хранится в счётчике **MaxMatrixCols**, как его можно изменять, становится ясно из следующего примера:

```
\[\setcounter{MaxMatrixCols}{15} \begin{matrix}
&&&&& 1\\&&&& 1 && 1\\ &&& 1 && 2 && 1\\
 && 1 && 3 && 3 && 1\\& 1 && 4 && 6 && 4 && 1 \\
1 && 5 && 10 && 10 && 5 && 1\\   \end{matrix} \]
```

$$\begin{matrix}
&&&&& 1\\&&&& 1 && 1\\ &&& 1 && 2 && 1\\
 && 1 && 3 && 3 && 1\\& 1 && 4 && 6 && 4 && 1 \\
1 && 5 && 10 && 10 && 5 && 1\\   \end{matrix}$$

После ввода большой матрицы значение счётчика лучше вернуть назад, чтобы не расходовать впустую ресурсы TEX'а.

Окружение `smallmatrix` позволяет набирать маленькие матрицы и перестановки, включённые в строку текста:

Матрица вида $\left(\begin{smallmatrix}a&b\\c&d\end{smallmatrix}\right)$ еще может поместиться внутрь абзаца, такая как $\left(\begin{smallmatrix}1&3&2&4\\3&f(x)&1&2\end{smallmatrix}\right)$ тоже смотрится неплохо, но громоздкая матрица наподобие

$$\mathbf{A} = \left(\begin{matrix} \dfrac{\varphi\cdot X_{n,1}}{\varphi_1\times\varepsilon_1} & (x+\varepsilon_2)^2 & \cdots & (x+\varepsilon_{n-1})^{n-1} & (x+\varepsilon_n)^n \\ \dfrac{\varphi\cdot X_{n,1}}{\varphi_2\times\varepsilon_1} & \dfrac{\varphi\cdot X_{n,2}}{\varphi_2\times\varepsilon_2} & \cdots & (x+\varepsilon_{n-1})^{n-1} & (x+\varepsilon_n)^n \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \dfrac{\varphi\cdot X_{n,1}}{\varphi_n\times\varepsilon_1} & \dfrac{\varphi\cdot X_{n,2}}{\varphi_n\times\varepsilon_2} & \cdots & \dfrac{\varphi\cdot X_{n,n-1}}{\varphi_n\times\varepsilon_{n-1}} & \dfrac{\varphi\cdot X_{n,n}}{\varphi_n\times\varepsilon_n} \end{matrix}\right) + \mathbf{I}_n$$

внутри абзаца выглядит неуместно, поэтому мы выключили ее из текста.

В этом примере матрицы, включенные в строку текста, набраны следующим образом:

```
$\left(
\begin{smallmatrix}a&b\\c&d\end{smallmatrix}
\right)$
$\left(
\begin{smallmatrix}1&3&2&4\\3&f(x)&1&2\end{smallmatrix}
\right)$,
```

громоздкую матрицу легко набрать описанными выше средствами, если подключить пакеты `amsmath` для создания отточия и `amssymb` для некоторых шрифтов.

3. Коммутативные диаграммы

3.1. Возможности окружения array

Окружение array можно использовать не только для верстки матриц: фактически оно создает таблицы, состоящие из строк и столбцов, например:

```
\[ \begin{array}{ccccc}
&&\mathcal{O}&&\\&&\vline&&\\\mathcal{M}&\gets&XY
&\to&\mathcal{N}\\&&\vline&&\\&&\infty&&\\
\end{array} \]
```

$$\begin{array}{ccccc}
&&\mathcal{O}&&\\&&\vline&&\\\mathcal{M}&\gets&XY
&\to&\mathcal{N}\\&&\vline&&\\&&\infty&&\\
\end{array}$$

Если какая-то графа в таблице должна быть пуста, то между (или перед, если эта графа — первая в своей строке) соответствующими знаками & нужно просто ничего не писать (или оставить сколько угодно пробелов). Если после записи в строке, до конца строки идут только пустые графы, то можно не дописывать до конца значки &, а сразу написать \\.

3.2. Возможности пакета amscd

Пакет amscd из коллекции $\mathcal{A}\mathcal{M}\mathcal{S}$-LaTeX'а позволяет легко создавать диаграммы без диагональных стрелок. Для этого служит окружение CD, внутри которого команды @>>>, @<<<, @VVV и @AAA задают, соответственно, правую стрелку, левую стрелку, стрелку вниз и стрелку вверх.

```
\[\begin{CD} S^{W_\Lambda}\otimes T@>>>T\\@VVV
@VVV\\(S\otimes T)/I@=(Z\otimes T)/J  \end{CD}\]
```

$$\begin{CD}
S^{W_\Lambda}\otimes T @>j>> T\\
@VVV @VVP\\
(S\otimes T)/I @= (Z\otimes T)/J
\end{CD}$$

Команды `@>>>`, `@<<<`, `@VVV` и `@AAA`, рисующие стрелки, могут разрываться надписями. Для горизонтальных стрелок, запись между первым и вторым символами > или < печатается в виде верхнего индекса, а запись между вторым и третьим символами — в виде нижнего индекса. Аналогично, запись между первым и вторым или вторым и третьим символами A или V вертикальных стрелок будет печататься как левый или правый «боковой индекс». Дополним изображенную выше диаграмму обозначениями отображений:

```
\[\begin{CD}
S^{W_\Lambda}\otimes T@>j>>T\\
@VVV@VV{  P}V\\
(S\otimes T)/I @= (Z\otimes T)/J
\end{CD} \]
```

$$\begin{CD}
X @>A>> Y\\ @V{B_1,\dots,B_p}VV @AA{C}A \\
R^p @>>{F(B_1,\dots,B_p)}> R
\end{CD}$$

Вертикальные стрелки могут иметь разные направления:

```
\[ \begin{CD}
X @>A>> Y\\ @V{B_1,\dots,B_p}VV @AA{C}A \\
R^p @>>{F(B_1,\dots,B_p)}> R
\end{CD}\]
```

$$\begin{diagram}
\node{\boxed{X}}
\arrow[2]{e,t}{A}
\arrow{s,l}{B_1,\dots,B_p}
\node[2]{Y} \\
\node{R^p}
\arrow[2]{e,b}{F(B_1,\dots,B_p)}
\node[2]{R} \arrow{n,r}{C}
\end{diagram}$$

3.3. Примеры использования пакета pb-diagram

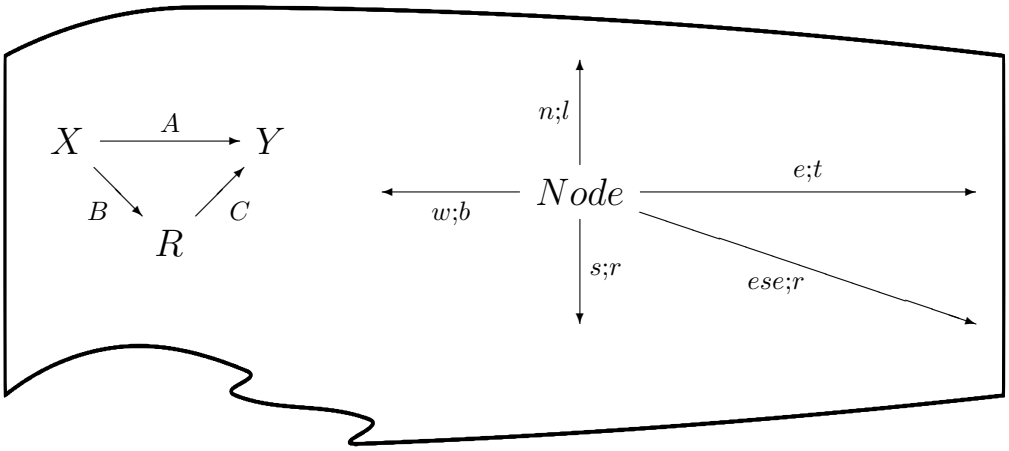
Пакет `pb-diagram` является более мощным, чем `amscd`, он позволяет рисовать наклонные стрелки. Кроме того, с помощью этого пакета более качественно изображаются вертикальные и горизонтальные стрелки (горизонтальные стрелки получаются равной ширины). Убедимся в этом, набрав предыдущий пример средствами пакета `pb-diagram`:

```
\[ \begin{diagram}
\node{\boxed{X}}
\arrow[2]{e,t}{A}
\arrow{s,l}{B_1,\dots,B_p}
\node[2]{Y} \\
\node{R^p}
\arrow[2]{e,b}{F(B_1,\dots,B_p)}
\node[2]{R} \arrow{n,r}{C}
\end{diagram}\]
```

$$\begin{diagram}
\node{\boxed{X}}
\arrow[2]{e,t}{A}
\arrow{s,l}{B_1,\dots,B_p}
\node[2]{Y} \\
\node{R^p}
\arrow[2]{e,b}{F(B_1,\dots,B_p)}
\node[2]{R} \arrow{n,r}{C}
\end{diagram}$$

В команде `arrow` направления стрелок кодируются буквами `n`, `e`, `w`, `s`, а положение надписи — буквами `t`, `b`, `l`, `r`. Необязательный аргумент `[k]` удлиняет стрелку в `k` раз.

```
\[ \begin{diagram}
\node{X}\arrow[2]{e,t}{A}
\arrow{se,b}{B}\node[2]{Y}\\
\node[2]{R}\arrow{ne,r}{C}
    \end{diagram}\qquad\qquad\qquad
     \begin{diagram}
\node{Node}\arrow{n,l}{n;l}\arrow{w,b}{w;b}
\arrow[2]{e,t}{e;t}\arrow{s,r}{s;r}
\arrow{ese,r}{ese;r}
     \end{diagram} \]
```



Более мощные средства для построения диаграмм содержат пакеты `xypic` и `pstricks`. Для `xypic` требуются специальные шрифты со стрелками (включены в дистрибутив пакета) или драйвер `PostScript`'а. Пакет `pstricks` работает только с `PostScript`'ом.

4. Окружения типа теоремы

Математические тексты содержат теоремы, леммы, определения, аксиомы и т.д. LaTeX позволяет организовать нумерацию подобных структур таки чтобы номера и ссылки на них создавались автоматически.

Для этого используется команда `\newtheorem`, которая создает окружения для набора различного вида «теорем». Новые окружения объявляются в преамбуле документа.

```
\newtheorem{<окружение>}[<счетчик>]%
                {<название>}[<раздел>]
```

Команда `\newtheorem` обычно употребляется в одном из следующих двух вариантов:

```
\newtheorem{<окружение>}[<счетчик>]{<название>}
\newtheorem{<окружение>}{<название>}[<раздел>]
```

Команда имеет два обязательных аргумента.

Аргумент `<окружение>` — это название нового типа окружения, — краткое ключевое слово, используемое для идентификации теоремы. Оно не должно совпадать с именем уже существующего окружения или счетчика. Аргумент `<название>` определяет то название окружения типа теоремы (теорема, лемма, определение, закон), под которым она будет печататься в документе. По умолчанию он печатается полужирным шрифтом, а формулировка — курсивом. Абзац, идущий после окружения начинается с абзацным отступом, если после закрывающей окружение команды идет пустая строка, и без отступа в противном случае.

Аргументы в квадратных скобках необязательны. Оба они используются для определения стиля нумерации теоремы.

Аргументом `<счетчик>` является название предварительно объявленной теоремы. При наличии этого аргумента для нумерации будет использоваться определенный ранее счетчик (например, счетчик `theorem`), тогда `<окружение>` и `theorem` будут иметь единую нумерацию. Новая теорема будет тогда нумероваться в той же последовательности. Если в списке аргументов `<счетчик>` отсутствует, то для каждого `<окружение>` создается счетчик с тем же именем.

Аргумент `<раздел>` позволяет указать раздел, внутри которого нумеруется создаваемое окружение. В варианте команды `\newtheorem{<окружение>}{<название>}[<раздел>]` аргумент `<раздел>` — это имя уже существующего счетчика, обычно счетчика команд секционирования. Фактически выбирается раздел, внутри которого будут нумероваться «теоремы». Создаваемый счетчик `<окружение>` является внутренним для `<раздел>`: он будет автоматически обнуляться при каждом изменении значения счетчика `<раздел>`.

Создаваемые окружения могут иметь один необязательный аргумент,

который обычно используется для указания автора; он будет печататься после номера «теоремы».

Ссылки на теоремы осуществляются также, как и ссылки на окружение типа `equation`.

Для следующих примеров новые «теоремы» созданы в преамбуле документа:

```
\newtheorem{theorems}{Теорема}
\newtheorem{tttt}{Интересная теорема}[section]
\newtheorem{laws}{Закон}[section]
\newtheorem{ddef}{Определение}
\newtheorem{jury}[ddef]{Нечто}
```

1 Первый раздел

2 Второй раздел

3 Третий раздел

Интересная теорема 3.1 (нумерация подчинена разделу)**.** *Это интересная теорема.*

Теорема 1 (Fermat)**.** *Не существует целых чисел $n > 2$, x, y и z таких, что $x^n + y^n = z^n$.*

Теорема 2. *Любой линейный оператор $\boldsymbol{A}$ имеет единственный сопряженный.*

Закон 3.1. *Если существует два или более способа сделать нечто, и один их этих способов может привести к катастрофе, то кто-то обязательно это сделает.*

Закон 3.1 установил Мчрфи.

4 Четвертый раздел

Интересная теорема 4.1 (нумерация подчинена разделу). *Еще одна интересная теорема.*

Теорема 3. *Теорема 1 верна.*

Закон 4.1 (Сенека). *Переноси с достоинством то, что изменить не можешь.*

Определение 1. *Кто знает, не говорит.*

Нечто 2 (Сервантес). *Осла медом не кормят!*

Определение 3. *Кто говорит — не знает.*

Исходный текст выглядит следующим образом:

```
\section{Третий раздел}\begin{tttt}[нумерация
подчинена разделу] Это интересная теорема.
\end{tttt}
\begin{theorems}[Fermat]\label{law:box}
Не существует целых чисел $n>2$, $x$, $y$ и $z$
таких, что $x^n+y^n=z^n$.      \end{theorems}
\begin{theorems}
Любой линейный оператор $\boldsymbol A$
имеет единственный сопряженный.\end{theorems}
\begin{laws}\label{Murphy}
Если существует два или более способа сделать нечто,
и один их этих способов может привести к катастрофе,
то кто-то обязательно это сделает. \end{laws}
Закон~\ref{Murphy} установил Мерфи.
```

```
\section{Четветрый раздел}
\begin{tttt}[нумерация подчинена разделу]
Еще одна интересная теорема.
\end{tttt}
\begin{theorems}
Теорема \label{law:box} верна.
\end{theorems}
\begin{laws}[Сенека]
Переноси с достоинством то, что изменить не можешь.
\end{laws}
\begin{ddef}
Кто знает, не говорит.
\end{ddef}
\begin{jury}[Сервантес]\emph{}
Осла медом не кормят!
\end{jury}
\begin{ddef}Кто говорит --- не знает.
\end{ddef}
```

Окружение `theorems` (заголовок «Теорема») определяет сплошную нумерацию. Счетчики окружений `tttt` и `laws` подчинены счетчику текущего раздела, но от друг от друга независимы. В части 4 создается окружение `ddef` и ему присваивается заголовок «Определение». В каждой секции окружения `ddef` будут нумероваться независимо друг от друга. Окружение `jury` (заголовок «Нечто») использует тот же счетчик, что и теорема `ddef`. Следовательно, она получает номер в последовательности с другими теоремами `ddef`. Аргумент в квадратных скобках указывает заголовок теоремы, или нечто аналогичное. Теорема `laws` (заголовок «Закон») получает номер, связанный с номером текущего раздела. Можно также использовать другую структурную единицу, например, главу или подраздел.

Пакет `amsthm` из коллекции $\mathcal{A}\mathcal{M}\mathcal{S}$-LaTeX'а добавляет декларацию `\newtheorem*{<окружение>}{<название>}` для печати теорем без номера и окружение `proof` для печати доказательств.

Лекция 6 ⏮

Библиография и цитирование литературы

Создание списка литературы. Библиографические базы данных. Программа BibTeX.

Создание списка цитируемой литературы и оформление на него ссылок является, пожалуй, самым трудоемким из технических процедур подготовки научной статьи и объемных научных трудов.

В LaTeX'е более-менее автоматизированно это можно сделать двумя способами.

1. Оформление ссылок на литературу в исходном файле

Ссылки на литературу можно создать при помощи окружения `\begin{thebibliography}...\end{thebibliography}`. В серьезных научных трудах библиография оформляется в виде отдельной главы:

```
\addcontentsline{toc}{chapter}{Литература}
\begin{thebibliography}{99}
\bibitem{metkacite} выходные данные публикации 1.
    <...>
\bibitem{metkacite} выходные данные публикации  n.
```

Конструкция означает, что глава «Литература» добавляется в конец списка оглавления, а ссылки (`bibitem`) указываются в порядке перечисления их ниже.

Рассмотрим в качестве примера документ класса `article`, в котором ссылки на литературу нумеруются в порядке упоминания.

Исходный код нужного нам фрагмента документа выглядит следующим образом (использован пример М. Конника на http://mydebianblog.blogspot.com):

```
\documentclass[a4paper,12pt,oneside]{article}
\usepackage[T2A]{fontenc}
\usepackage[cp1251]{inputenc}
\usepackage[english,russian]{babel}
\usepackage{amssymb,amsfonts,amsmath,
mathtext,cite,enumerate,float}
\begin{document}
<...>
Метод Голда\cite{gold} получен из метода
ван Циттерта\cite{gold-cittert},
но с предположением, что действует он
только для положительных чисел.
<...>
\begin{thebibliography}{8}% Список литературы
\bibitem{blind}
S. M. Jefferies and J. C. Christou.
Restoration of astronomical images by iterative
blind deconvolution.
Astrophysical Journal, 415:862+, October 1993.
<...>
\bibitem{gold}
GOLD, R.: An Iterative Unfolding Method for
Matrices, Tech. Rep. ANL-6984,
Argonne National Laboratory, Argonne, Illinois, 1964.
\bibitem{gold-cittert}
Journal of Electrical Engineering,
Vol. 56, NO. 5-6, 2005, 141--145
Multidimensional FFT Based Positive
Definite Gold Deconvolution Algorithm
Miroslav Morh\'a\v{c} -- Vladislav Matou\v{s}ek
<...>
\end{thebibliography}
\end{document}
```

В этом случае ссылки `\cite{metkacite}` будут пронумерованы в том порядке, в котором они упомянуты в окружении `\begin{thebibliography}` ... `\end{thebibliography}`, но не в порядке упоминания в тексте.

Полученный после компиляции документ (для создания ссылок требуется два раза обработать документ при помощи `latex` или `pdflatex`) будет выглядеть следующим образом:

«...»

Метод Голда [4] получен из метода ван Циттерта [5], но с предположением, чт
чисел.

«...»

Список литературы

[1] S. M. Jefferies and J. C. Christou. Restoration of astronomical images by
Journal, 415:862+, October 1993.

[2] Van CITTERT, P. H. : Zum Einfluss der Spaltbreite auf die Intensittverteilu
69 (1933), 298.

[3] N.R. Hill, G.E. Ioup: J. Opt. Soc. Am. 66 (1976) 487

[4] GOLD, R. : An Iterative Unfolding Method for Matrices, Tech. Rep. ANL-69
Illinois, 1964.

[5] Journal of Electrical Engineering, Vol. 56, NO. 5-6, 2005, 141–145 Multidin
Deconvolution Algorithm Miroslav Morháč – Vladislav Matoušek

[6] L.B. Lucy. An iteration technique for the rectification of observed distributic

Заметим, что если при работе с большими списками цитируемой литературы случайно переставить местами в окружении `\begin{thebibliography}` ... `\end{thebibliography}` ссылки `\bibitem{gold}` и `\bibitem{gold-cittert}`, то в тексте ссылки будут стоять так:

Метод Голда [5] получен из метода ван Циттерта [4], но с предположением, чт
чисел.

Такая особенность затрудняет работу с большими документами, а также с документами, составленными из нескольких файлов. Особенно ярко этот недостаток проявляется тогда, когда часть документа с цитированием литературных источников вставляется в другой документ.

2. Создание списка литературы с использованием библиографической базы данных BiBTeX

Сократить время формирования списка и избежать указанных трудностей позволяет создание собственной, периодически обновляемой библиографической базы данных автора. Источники литературы в этом случае добавляются не в сам LaTeX-документ, а во вспомогательный `bib`-файл, что позволяет использовать созданную библиографическую базу для многих документов.

Библиографические базы имеют определенную структуру. База хранится в обычном текстовом файле с расширением `.bib` (в нем перечисляются все библиографические ссылки автора). В процессе работы с файлом `имяфайла.tex` программа `BibTex` извлекает из базы данных источники литературы, указанные в тексте файла.

Преимущества использования библиографических баз данных

1. Ссылки в основном документе появляются по мере упоминания и всегда нумеруются правильно.

2. Существует возможность использования одной базы во многих документах.

3. Если в описание ссылки на литературу вкралась ошибка, её можно исправить в базе, тем самыи она будет исправлена сразу во всех ссылающихся на базу документах.

4. Существует простой поиск в базе для выяснения ключа цитирования (аргумента `\cite{metkacite}`).

5. Возможен импорт и экспорт в другие библиографические форматы.

Порядок формирования и работы с личной базой данных

1) Наличие необходимых файлов с расширениями `.bib` и `.bst`.

Как уже отмечалось, сама база данных находится в файле с расширением `.bib`. Перед началом необходимо создать и заполнить файл с таким расширением.

Каждая запись должна быть отнесена к определенному типу: статья в журнале, статья для конференции, книга, дипломная работа и так далее.

Все записи файла `имяфайла.bbl` должны иметь определенную структуру. Записи различных типов в базе данных имеют различный набор полей, причем поля делятся на 3 класса: обязательное поле, необязательное поле и игнорируемое поле. Подробно ознакомиться со списком полей для различных типов записей можно в документации по BiBTeXу.

Данные можно набивать вручную в любом текстовом редакторе, а можно поставить программу-оболочку для работы с BibTeXом (например, бесплатную программу `JabRef`, написанную на `Java`).

Записи в Файле с расширением bib будет иметь, например, следующий вид:

```
@ARTICLE{catheydowskinewparadigm,
author = {W.T. Cathey and E.R. Dowski},
title = {New paradigm for imaging systems},
journal = {Appl. Opt.},
year = {2002},
volume = {41},
pages = {6080-6092},
owner = {penta4},
timestamp = {2006.05.12},
}
```

Второй файл, необходимый для работы `BibTeX`, это стилевой файл с расширением `.bst`, определяющий библиографический формат.

В соответствии с библиографическим форматом будут формироваться ссылки и список литературы каждой конкретной статьи.

Файл с расширением `.bst` для формирования списка литературы нужного стиля можно взять в каталоге `.bst`-файлов на `Graham Williams' LaTeX Catalogue Online, CTAN Edition`. Многие научные издательства имеют свой `.bst`, который отражает конкретные требования издательства к оформлению литературы.

2) Команды в основном файле статьи (или другого научного труда).

В преамбуле статьи нужно разместить команду `\bibliographystyle{библиографический стиль}`, аргумент которой — это имя используемого `.bst`-файла `библиографический стиль.bst`.

Существует четыре стандартных стиля: *plain, unsrt, alpha, abbrv*. С коллекцией AmSLaTeX'а распостраняются еще два стиля: *amsaipha* и *amsplain*.

В конце статьи вместо окружения `\thebibliography` нужно разместить команду `\bibliography{имя файла базы данных}`, где имя файла указывается без расширения.

Например, в рассмотренном выше примере вместо окружения `\begin{thebibliography} ...\end{thebibliography}` надо использовать `\bibliography{my}`, если база называется `my.bib` (расширение не указывается). Кроме того, в преамбулу добавляется несортированный стиль библиографии `\bibliographystyle{unsrt}` (номера — в порядке упоминания в тексте). В результате «на выходе» внешний вид файла не изменится.

3) Последовательность действий при работе с подготовленным файлом статьи и базой данных.

- Статья обрабатывается один раз при помощи latex (pdflatex). В результате этого в файле с расширением `.aux` генерируются `\cite` ссылки статьи.

- В командной строке (например в `Far`) дается команда `bibtex имястатьи`, где имя файла статьи указывается без расширения. При этом, на основании анализа файла `.aux`, `BibTeX` создает файл `.bbl`, содержащий ссылки, отформатированные в соответствии с библиографическим форматом, определенном в стилевом файле `.bst`. Предупреждения и сообщения об ошибках записываются в файл `.blg`.

- Теперь LaTeXдолжен считать файл `.bbl` (статья еще раз обрабатывается при помощи latex).

- Чтобы ссылки в тексте приняли окончательный вид, статья третий раз обрабатывается при помощи latex (pdflatex).

3. Полезные советы

1. Иногда требуется оформлять номера списка литературы через точку (1. 2. ... N.). Для этого в преамбуле файла статьи до `\begin{document}` нужно указать

```
\makeatletter
\bibliographystyle{unsrt}
\renewcommand{\@biblabel}[1]{#1.}
\makeatother
```

2. В одном документе можно использовать не одну базу, а несколько. В этом случае имена файлов баз указываются через запятую без пробелов, например так: `\bibliography{biblio/my,biblio/your}`. Здесь `biblio` — имя папки в рабочей директории, в которой хранятся соответствующие файлы.

3. Некоторые научные журналы требуют явно вставлять библиографические ссылки в текст статьи в окружении `\thebibliography`. Такой файл легко сделать на основе имеющегося файла статьи со ссылкой на базу данных — в копию файла статьи вставляется содержимое файла `.bbl` и заменяются соответствующие команды.

Лекция 7 ⏮

Управление заголовками. Создание указателей. Стиль автоматически формируемого содержания

Синтаксис описания элементов указателя. Подготовка указателя. Программа MakeIndex.

1. Управление заголовками

1.1. Построение алфавитного указателя

В этой разделе мы рассмотрим составление алфавитного указателя. Уже из его названия ясно, что в таком указателе список терминов упорядочен по алфавиту. LaTeX предоставляет средства, позволяющие полностью автоматизировать составление алфавитного указателя и значительно облегчить подготовку словаря терминов. Текст алфавитного (равно как и любого другого) указателя печатает процедура `theindex`.

Подготовка алфавитного указателя — весьма длительная процедура. В качестве первого шага в исходном тексте следует пометить выбранные термины при помощи команды `\index`. Если некоторые термины встречаются в тексте многократно, то возможны варианты. Например, можно пометить только первое упоминание каждого термина. Другая крайность — механически отметить каждое упоминание термина. Выбор варианта целиком определяется автором публикации. Обычно отмечают наиболее информативные упоминания каждого термина, но, например, в научных публикациях по истории часто фиксируется каждое упоминание события.

Чтобы пометить термин в исходном тексте документа в простейшем случае его достаточно продублировать в аргументе команды `\index`, которую размещают сразу вслед за термином.

На завершающей стадии подготовки указателя в корневой входной файл (допустим, он называется `my.tex`) нужно внести следующие изменения: 2. 3. 4.

1. Загрузить пакет `makeidx` и вставить в преамбулу декларацию `\makeindex`. Она указывает, что LaTeX должен обновлять перечень помеченных терминов при очередной компиляции исходного текста.

```
\documentclass{book}
\usepackage[russian]{babel}
\usepackage{makeidx}
\makeindex
. . .
\begin{document}
. . .
```

2. Вставить команду `\printindex` там, где должен быть напечатан алфавитный указатель. Обычно её помещают ближе к концу документа.

3. Повторить компиляцию входного файла. В результате будет создан первичный указатель `my.idx`. Он состоит из входов в указатель, каждый из которых создан одной командой `\index` в исходном тексте.

4. Отсортировать первичный указатель программой `MakeIndex`. В результате будет создан файл my.ind, содержащий исходный текст алфавитного указателя в виде процедуры `theindex`.

При следующей компиляции исходного текста команда `\printindex` напечатает алфавитный указатель, взяв его текст из `my.ind`.

Несколько команд `\index` с одним и тем же аргументом на одной странице не влияют на построение указателя, поэтому термин, который должен попасть в индекс, можно каждый раз вводить в сопровождении команды `\index` (например, не просто `tabular`, а `tabular\index{tabular}`).

Описанные шаги в построении алфавитного указателя позволят построить лишь простейший указатель — одноуровневый. Если речь идет

о построении многоуровневого указателя, то необходимо добавить, что аргумент команды `\index` имеет несколько форматов, которые позволяют построить более сложные указатели.

```
\index{<вход>}
```

Вход в алфавитном указателе будет ссылаться на номера страниц.

```
\usage{index}{<вход>|(}
\usage{index}{<вход>|)}
```

Вход в алфавитном указателе будет ссылаться на область страниц.

```
\index{<вход>!<субвход>}
```

Вход в алфавитном указателе будет иметь субвход первого уровня.

```
\index{<вход>!<субвход>!<субсубвход>}
```

Вход в алфавитном указателе будет иметь субвход второго уровня.

```
\index{<вход1>!<субвход1>|see{<вход, субвход>}}
\index{<вход1>|see{<вход>}}
```

Вход в алфавитном указателе будет ссылаться на другой вход. Вход в алфавитном указателе будет иметь субвход второго уровня.

```
{\index}{<вход>|<имя команды>}
```

Чтобы ввести особый формат выдачи номера страницы (например, номер страницы, где термин встречается впервые, желательно выделить курсивом), в аргументе команды `\index` следует после разделителя `|` указать имя команды форматирования без символа `\`.

Чтобы поместить в аргумент команды `\index` символы `!`, `@`, `|` как обычные символы, перед каждым из них следует выдать символ двойных кавычек.

Замечание. Команды `\index`, аргументы которых состоят из одних и тех же символов, но по-разному разделенных пробелами, создают в алфавитном указателе различные входы.

2. Стиль автоматически формируемого содержания

2.1. Что нумеровать и что включать в оглавление

LaTeX определяет, какие разделы документа нумеровать, а какие — нет, исходя из двух вещей: *уровня вложенности раздела* и значения счетчика `secnumdepth`. Раздел документа получает номер, если уровень его вложенности меньше или равен значению `secnumdepth` (разумеется, все сказанное относится к случаю, когда LaTeX'овская команда для раздела документа дана без звездочки — иначе нумерации не будет заведомо). Уровни вложенности в стандартных стилях приведены в таблице.

Название раздела	Уровень
`\section`	1
`\subsection`	2
`\subsubsection`	3
`\paragraph`	4
`\subparagraph`	4

Стало быть, если мы хотим, чтобы самый мелкий из нумеруемых разделов был `\paragraph`, то надо написать в преамбуле документа команду

```
\ setcounter{secnumdepth}{3}
```

Что касается команд `\chapter` и `\part`, то соответствующие разделы документа будут нумероваться тогда и только тогда, когда значение `\secnumdepth` является неотрицательным числом.

Какие разделы включать в оглавление, определяется другим счетчиком, а именно `tocdepth`: информация о разделе документа вносится в оглавление в том и только том случае, если уровень вложенности раздела меньше или равен значения этого счетчика (и раздел вводится командой без звездочки).

Если мы напишем в преамбуле

```
\ setcounter{tocdepth}{3}
```

то в оглавление будут вынесены разделы документа на уровне параграфов.

Напомним, что *-форма команд секционирования не нумерует разделы и не передает заголовки разделов в оглавление и колонтитулы. Если мы оформили **Введение** в виде

```
\section*{Введение}
```

(а **Введение** обычно не нумеруют), то слово **Введение** в оглавлении не появится. Поэтому, чтобы название раздела появилось в оглавлении, необходимо предыдущий раздел документа необходимо задать в виде

```
\section*{Введение}
\addcontentsline{toc}{section}{Введение}.
```

2.2. Модификация оглавлений

Поясним как это происходит. Оглавление, списки рисунков и таблиц печатают соответствен- но команды

```
\tableofcontents
\listoffigures
\listoftables
```

Они считывают необходимую информацию из файлов, имена которых совпадают с именем входного файла (точнее, корневого входного файла), а расширения определяются из следующей таблицы:

```
команда:\tableofcontents  \listoffigures  \listoftables
расширение:  toc                lof             lot
```

Информация для оглавления формируется командами секционирования, а информация о рисунках и таблицах — командой `\caption` в процедурах `figure` и `table`. Кроме того, такую информацию можно записать с помощью следующих команд.

```
\addcontentsline{расшир}{раздел}{текст}
```

добавляет запись `текст` в файл с указанным расширением `расшир`, форматируя её в соответствии со значением аргумента `раздел`. Расширение имени файла `расшир` может принимать одно из трёх значений: `toc`,

lof, lot. В зависимости от конкретного выбора **расшир** второй аргумент **раздел** должен иметь следующие значения:

toc: part, chapter, section и т. д. (имя команды секционирования без \) lof: figure lot: table

Текст записи **текст** является подвижным аргументом. Чтобы в оглавление или список ввести запись с номером раздела, рисунка или таблицы, **текст** должен иметь следующую форму:

```
\protect\numberline{номер}{текст записи}
```

Для того, чтобы записать в toc- (соответственно, lof- или lot) файл любой текст и любые TEX'овские команды, служит следующая команда

```
\addtocontents{расшир}{текст}
```

Здесь **расшир** по-прежнему означает расширение имени файла, в который следует записать информацию (toc, lof, lot). Второй аргумент **текст** является подвижным и должен содержать текст записи.

Команды \addcontentsline и \addtocontents позволяют редактировать служебные файлы с расширением toc, lof и lot, если необходимо осуществить тонкую настройку, например при ручном регулировании линии переноса на новую страницу части длинной таблицы.

Вы хотите в каком-то месте оглавления провести горизонтальную линейку шириной во всю страницу, то можно написать

```
\addtocontents{toc}{\hrule}
```

и в оглавлении появится линейка.

Пусть Вы не хотите, чтобы какая-то из строк в оглавлении начинала новую страницу. Тогда надо перед командой, порождающей эту строку оглавления (обычно таковой будет команда наподобие \section) написать в своем файле:

```
\addtocontents{toc}{\protect\nopagebreak}
```

в toc-файл запишется команда , и нежелательный разрыв страницы в оглавлении будет предотвращен. Если забыть про \protect, то получили бы весьма непонятное сообщение об ошибке.

Если необходимо озаглавить по другому само оглавление, списки рисунков или таблиц, то необходимо переопределить команды, хранящие названия оглавления, списка рисунков и таблиц:

```
\contentsname
\listfigurename
\listtablename
```

Например, мы можем переопределить список фигур:

```
\renewcommand{\listfigurename}{Список графиков}
```

Раздел 2. Основы программирования в TEX и LATEX

Лекция 8 ⏮

TeX как язык программирования

Символы TEX, коды и категории

TEX читает строки исходного файла и преобразует их в поток элементов, из которых в последствии формируются строки выходного `dvi`-файла. Таким образом форматирование исходного файла практически не влияет на то, как будет выглядеть результат. Напротив, некоторые символы оказывают на процессы, происходящие внутри TEX, весьма существенное влияние. Это влияние зависит как от кода символа так и от его категории.

1. Категории символов

Все символы, с которыми сталкивается TEX в процессе обработки входного файла, подразделены на 16 категорий с номерами от 0 до 15.

Номер	*Значение*	*Символ по умолчанию*
0	Начало команды	\
1	Начало группы	{
2	Конец группы	}
3	Математический ключ	$
4	Табулятор	&
5	Конец строки	<return>
6	Параметр	#

Номер	*Значение*	*Символ по умолчанию*
7	Верхний индекс	`^`
8	Нижний индекс	`_`
9	Игнорируемый символ	`<null>`
10	Пробел	
11	Буква	`A-Z, a-z, A-Я, a-я`
12	Другой символ	`все, кроме перечисленных`
13	Активный символ	`~`
14	Комментарий	`%`
15	Ошибочный символ	`<delete>`

Категорию любого символа можно изменить, однако следует помнить, что форматы TEX используют некоторые символы для специальных целей. Если нужно использовать какой-либо из этих символов в тексте так, чтобы он появился в выходном файле, то для этого определены специальные команды, такие как

```
\$, \%, \&, \#, \_
```

в обычном режиме, и

```
\{, \}, \backslash
```

в математическом. Команды `\^` и `\~` производят акценты над буквами.

Когда TEX читает строку текста из файла, он преобразует этот текст в список элементов. Элемент — это либо символ с номером категории, либо команда. Команда рассматривается как один объект, который больше не является последовательностью символов. Пробелы после команд в списке элементов не игнорируются так, как это делается во входном файле. Прикрепленныек к элементам номера категорий не будут изменяться, даже если категория соответствующих символов во входном файле будет изменена. Изменить категорию символа можно командой

```
\catcode`\<символ>=<число>
```

В результате символу **<символ>** будет присвоен номер категории **<число>**. Как мы увидим далее, сочетание `` `\ `` имеет специальное значение, и заменяется кодом следующего за ней символа.

Не смотря на то, что команды рассматриваются как цельные элементы, существует возможность преобразовать их в список символьных элементов, используя

```
\strings\<команда>
```

где **<команда>** — это имя какой-либо команды.

В результате образуется список элементов, состоящий из символа обратной косой черты и из символов, входящих в имя команды. Каждый из символов в таком списке получает номер категории 12 (другие). Исключение будет сделано только для символа пробела, его категория останется прежней (категория 10), однако представить ситуацию когда пробел сможет появиться в имени команды не просто.

Перейти в обратную сторону, от списка символьных элементов к элементу-команде можно посредством

```
\csname<элементы>\endcsname
```

<элементы> могут в том числе включать и другие команды, однако только такие, которые будут заменены на символы, а не на примитивы TEX. О том, каким образом обрабатываются команды, пойдет речь ниже.

Еще две команды, производящие элементы, это команды

```
\number<число>
\romannumeral<число>
```

которые заменяют **<число>** на его эквивалент в десятичной или римской системе счисления соответственно.

```
\uppercase<список элементов>
\lowercase<список элементов>
```

преобразуют **<список элементов>** в верний или нижний регистр соответственно. Осуществляется это засчет того, что у каждого символа кроме его кода и категории существует еще пара кодов — `\uccode` и `\lccode`.

При переводе символа в верхний регистр его код заменяется на `\uccode`, при переводе в нижний — на `\lccode`, если только результирующий код не равен нулю. В противном случае код символа остается без изменений. Отметим, что эти команды обрабатывают элементы, а не символы в исходном файле.

2. Коды символов

Во входном файле TEX встречает массу символов. Каждому из них поставлен в соответствие его код при помощи кодовой таблицы. Для англоязычных стран обычно это будет кодовая таблица ASCII. В этой таблице символам сопоставлены числа от 0 до 127. В эти 128 символов попадают как все печатные, так и некоторые непечатные, имеющие специальное значение. В странах, использующих наряду с английским свой национальный алфавит используется расширенная таблица ASCII, символам национального алфавита отведены коды со 128 до 255.

Кодировка, которую использует TEX для внутреннего представления символов, также основана на ASCII, и шрифты TEX устроены так, что коды тех символов, которые есть в ACSII, остаются такими же и в шрифте. Однако шрифт содержит 256 символов, тогда как отображаемых символов ASCII заведомо меньше. Кроме того, многие из символов отсутствуют на клавиатуре. Для того, чтобы в выходном документе появился такой символ, используется команда `\char`, которая позволяет обращаться к символам шрифта по их кодам. Формат вызова этой команды такой:

```
\char<число>
```

где `<число>` — это любое число от 0 до 255 с необязательным пробелом после него. `<число>` может быть записано как в привычной десятичной системе счисления, так и в восьмеричной или шестнадцатеричной. Чтобы указать TEX что число нужно рассматривать как восьмеричное, перед ним ставится одинарная кавычка, например `'37`, перед шестнадцатеричным числом ставится двойная кавычка, `"5A`.

TEX предусматривает и неявное задание аргумента `<число>`, что избавляет от необходимости вообще знать коды символов. Элемент `` ` `` (левая

кавычка), если за ним следует символ TEX или команда, имя которой одиночный символ, означает внутренний код TEX для этого символа. Таким образом, определения `\char98`, `\char`b` и `\char`\b` будут эквивалентны.

Хотя `\char` может использоваться для доступа к любому символу шрифта, его, конечно, нельзя использовать внутри имени команд. Во время ввода текста автору вряд ли придется прибегать к помощи команды `\char`, поскольку все необходимые для ввода символы наверняка уже будут описаны в виде команд в форматном или стилевом файле, однако для разработчиков форматов или стилей эта команда необходима. Альтернативой ей при определении команд может стать команда

```
\chardef\<команда>=<число>
```

где `<команда>` определяет имя новой команды, а `<число>` задает код символа одним из перечисленных выше способов.

Как уже было замечено ранее, количество символов, которые можно ввести с клавиатуры, заметно меньше количества символов в шрифте. Для ввода невидимых символов TEX имеет специальный способ. Это последовательность из трех видимых символов

```
^^<символ>
```

Если код символа `<символ>` меньше 64, то к нему прибавляется 64, если больше — то 64 из него вычитается. Таким образом, тройка `^^@` будет иметь код 0, `^^A` будет иметь код 1 и так далее. Запись символов при помощи `^^` отличается от `\char`, поскольку комбинация `^^` воспринимается как один символ, и может быть использована даже как буква в именах команд.

3. Правила чтения исходного файла

Для более глубокого понимания процессов, происходящих при обработке исходного файла, необходимо рассмотреть правила, которых TEX придерживается при чтении исходного файла.

Когда TEX читает очередную строку входного текста, он находится в одном из трех состояний:

- **N** начало новой строки;
- **M** середина строки;
- **S** пропуск пробелов.

В начале строки TEX будет в состоянии `N`, после команды или пробела в состоянии `S`, все остальное время в состоянии `M`. TEX удаляет все пробелы (символ с кодом 32) на правом конце входной строки, и вставляет символ с кодом 13 (перевод строки).

Если TEX в любом состоянии встречает символ начала команды (категория 0), он пытается определить имя команды следующим образом:

a) если в строке больше нет символов, то имя пусто;

b) если следующий символ не буква (категория не равна 11), то имя состоит из одного этого символа;

c) если следующий символ буква, то имя состоит из этого символа и из всех последующих, пока они являются буквами;

TEX переходит в состоянии S в случае 3 и в случае 2, если символ является пробелом (категория 10). В остальных случаях TEX переходит в состояние M.

Если TEX в любом состоянии встречает пару идущих подряд символов верхнего индекса ^ (категория 7), причем за ними следует не конец строки, то они удаляются, и к коду следующего за ними символа прибавляется (если код меньше 64) или вычитается число 64. После чего TEX продолжает работать так, как будто на этом месте всегда стоял символ с новым вычисленным кодом.

Если TEX встречает:

- символ категории 1, 2 ,3, 4, 6, 8, 11, 12, 13 или символ категории 7 за которым не следует еще один символ категории 7 (см. предыдущий абзац), то он преобразует этот символ в элемент, присваивая ему номер категории, и переходит в состояние M.

- символ конца строки (категория 5), то чтение строки на этом завершается, и все идущие следом символы, если они есть, отбрасываются. Далее, если TEX находился в состоянии N, то символ конца строки преобразуется в конец абзаца. Если TEX находился в состоянии M, то символ конца строки преобразуется в пробел (категория 10), а если TEX находился в состоянии S, то символ конца строки отбрасывается.

- игнорируемый символ (категория 9), то он пропускает его, ничего не делая.

- пробел (категория 10), то в состоянии N или S символ просто пропускается, а в состоянии M символ преобразуется в элемент пробел (категоря 10) и TEX переходит в состояние S.

- символ комментария (категория 14), он отбрасывает этот символ и все последующие символы в этой строке.

- ошибочный символ (категория 15), он печатает сообщения об ошибке и остается в том состоянии, в котором находился.

Если TEX не находит больше символов в текущей строке, он переходит к следующей строке и переходит в состояние N.

Лекция 9 ⏮

Счетчики, длины, боксы

Основные счетчики, команды для манипулирования со счетчиками, создание новых счетчиков; абсолютные и относительные единицы измерения, растяжимые длины, команды для манипулирования с длинами; боксы в TEX, команды работы с боксами, хранение текста в боксе, поднятие и опускание бокса.

В этой лекции будут даны краткие сведения о том, каким образом TEX формирует выходной файл.

1. Размеры

Для ручного управления относительным размещением объектов на странице иногда необходимо указывать расстояния между ними. Измеряя расстояния, TEX использует понятие размера. Это понятие пришло из традиционной типографии, и для задания размеров используются специальные единицы измерения, принятые в издательстве. Однако размеры можно задать и в метрической системе. Используемые в TEX единицы измерения следующие:

`pt`	point	пункт
`pc`	pica	пика (1 pc = 12 pt)
`in`	inch	дюйм (1 in = 72.27 pt)
`bp`	big point	большой пункт (72 bp = 1 in)
`cm`	centimeter	сантиметр (2.54 cm = 1 in)
`mm`	millimeter	миллиметр (10 mm = 1 cm)
`dd`	didot point	дидот-пункт (1157 dd = 1238 pt)
`cc`	cicero	цицеро (1 cc = 12 dd)
`sp`	scaled point	суперпункт (65536 sp = 1 pt)

Необходимо всегда учитывать, что размер для TEX это не просто число, указывая размер обязательно нужно указывать и единицы измере-

ния, даже если размер равен нулю. В общем случае размер задается так: `[<знак>]<число><единица измерения>`.

Необязательный параметр `<знак>` используется для задания отрицательных размеров, `<число>` может быть целым или дробным и задает числовое значение размера, `<единица измерения>` одна из перечисленных выше. Максимальное абсолютное значение размера составляет 2^30 sp, т.е. около 6 метров.

TEX измеряет все расстояния именно размерами, большую часть из которых задавать явно не приходится. Существует способ применить коэффициент растяжения (или сжатия) ко всем размерам в документе. Делается это с помощью команды

```
\magnification=<число>,
```

где `<число>` задает величину коэффициента, на который будут умножены все размеры, который ровно в 1000 раз меньше чем `<число>`. Т.е.

```
\magnification=1200
```

указывает растянуть все размеры в 1.2 раза. Конечно, чтобы TEX мог сформировать масштабированный документ в системе должны присутствовать шрифты подходящего размера.

После задания масштаба все размеры, в том числе и те, которые были заданы явно, будут изменены. Если необходимо задать размер, на который не будет влиять масштабирование, используйте ключевое слово `true` непосредственно перед единицей измерения размера.

Кроме описанных выше абсолютных размеров TEX имеет еще и размеры относительные, т.е. их точное значение зависит от контекста. Это:

em — ширина квадрата в текущем шрифте;

ex — высота буквы „x“ («икс») в текущем шрифте.

Каждый шрифт определяет собственные значения этих размеров.

TEX хранит размеры, которые использует при обработке исходного файла, в специальных внутренних регистрах, число которых 256, и называются они `\dimen0`...`\dimen255`. Регистр размера можно использовать во

всех командах вместо для указания значения размера. Чтобы дать регистру размера символическое имя, используйте команду

```
\ dimendef\<name>=<число>
```

где `<число>` задает номер регистра, а `<name>` определяет символическое имя, которое будет использоваться для доступа к регистру.

Определять собственные регистры размера можно командой

```
\ newdimen\<name>
```

После этого можно использовать `\<name>` как регистр размера. Создавая собственные регистры необходимо помнить, что фактически TEX не создает нового регистра, он выбирает один из 256, которые не используется в данный момент, и назначает ему символическое имя, как было показано выше.

С регистрами размера можно выполнять арифметические операции, используя команды

```
\advance \<name> by <размер>
\multiply\<name> by <размер>
\divide  \<name> by <размер>
```

Регистр `<name>` будет использован в качестве левого операнда, а `<размер>` в качестве правого для операции сложения, умножения или деления соответственно. Результат операции будет помещен в тот же регистр.

С точки зрения подхода к TEX как к языку программирования, регистры размера больше всего походят на типизованные переменные. В сущности, таковыми они и являются, наряду с регистрами счетчиков, клея и боксов, о которых еще пойдет речь.

2. Счетчики

Кроме регистров размеров, описанных выше, TEX имеет 256 числовых регистров, которые называются `\count0`...`\count255`. В этих регистрах хранятся целые числа. Они очень похожи на регистры размера, и отличаются от них только типом хранимых данных. Для определения символиче-

ских имен используется команда `\countdef`, для создания собственных числовых регистров используется `\newcount`.

TEX использует числовые регистры для автоматического подсчета количества различных объектов в процессе работы, таких, например, как страницы, главы, параграфы или формулы. Именно эти регистры используются для автоматической нумерации. Поэтому изменять их значения необходимо с осторожностью.

Числовые регистры можно использовать в арифметических операциях аналогично регистрам размера, за исключением того, что второй операнд должен быть числом.

Все, сказанное выше, относится в базовой системы TEX, в LATEX использовать прямой доступ к счетчикам нет необходимости.

Новый счетчик с именем `<name>` в LATEX создается командой

```
\newcounter{<name>}
```

Имя должно быть свободно, иначе возникнет сообщение об ошибке.

Чтобы задать новое значение счетчика, используется команда

```
\setcounter{<name>}{<число>}
```

которая устанавливает значение счетчика `<name>` равным значению `<число>`.

Другой способ изменить значение счетчика — прибавить к нему положительное или отрицательное число. Сделать это можно командой

```
\addcounter{<name>}{<число>}
```

Заметим, что команды, изменяющие значение счетчиков, являются глобальными.

Использовать значение счетчиков в тексте можно командами

```
\arabic{<name>}  \roman{<name>} \Roman{<name>}
```

Каждая из этих команд печатает текущее значение счетчика `<name>`. Первая использует для этого арабские цифры и десятичную систему счисления, вторая прописные латинские буквы и римскую систему счисления, третья отличается от второй тем, что буквы будут строчными.

Использовать значение счетчика `<name>` в вычислениях, а не для вывода, можно командой

```
\value{<name>}
```

Команды

```
\alph{<name>}        \Alph{<name>}
```

печатают строчную или прописную букву, которая стоит в алфавите на месте, номер которого равен значению счетчика `<name>`.

3. Боксы

Когда TEX имеет дело с потоком элементов (напомним, что поток элементов образуется из символов и команд исходного файла), то он ничего не знает о том, как они будут выглядеть на экране или листе бумаги. Все задачи по размещению элементов на странице решаются с использованием объектов, которые называются боксы (`box`).

Бокс — это двумерный объект прямоугольной формы, который измеряется тремя величинами, которые называются ширина, высота и глубина. Геометрически это прямоугольник, длина одной стороны которого равна ширине бокса, а длина другой равна сумме высоты и глубины бокса. Ширина измеряет бокс по горизонтали, а высота и глубина по вертикали. На расстоянии высоты бокса от верхней (соответственно на расстоянии глубины бокса от нижней) стороны прямоугольника проведен отрезок между боковыми сторонами, перпендикулярный им. Этот отрезок называется базовой линией бокса. Точка, находящаяся на пересечении базовой линии и левой стороны прямоугольника, называется точкой привязки.

TEX различает боксы двух типов — простые *символьные* и *линейчатые*. Символьный бокс ассоциирован с символом шрифта, и данные о его измерених хранятся в файле шрифта. *Линейчатые* боксы — это боксы, которые также имеют три измерения, однако они во время вывода будут целиком закрашены. Из таких боксов обычно делаются горизонтальные и вертикальные линейки.

Все, что размещается на странице, составляется из символьных и линейчатых боксов, которые склеиваются в различных комбинациях, гори-

зонтально или вертикально.

Когда боксы склеиваются горизонтально, точки привязки соседних боксов лежат на одной прямой, при создании вертикальных боксов они также располагаются на одной прямой, только уже вертикальной.

TEX может делать боксы, состоящие из боксов, и работать с ними как с простыми боксами. На этом основан алгоритм создания выходного документа. Сначала символьные и линейчатые боксы склеиваются горизонтально до тех пор, пока не будет набрана строка, которая сама становится боксом. Затем боксы-строки склеиваются вертикально, формируя более крупный бокс, которым является страница документа. В более сложных случаях вложенность боксов может быть и более глубокой.

Для создания горизонтального бокса используется команда `\hbox`, вертикального — `\vbox`.

Для хранения боксов TEX использует боксовые регистры `\box0`...`\box255`. Устанавливается значение регистра командой `\setbox`.

4. Клей

В отношении того, как TEX работает с боксами, часто использовалось слово «склеивает» для обозначения способа фиксации взаимного расположения боксов по вертикали и горизонтали. На самом деле, соединение боксов между собой происходит посредством объекта, который действительно называется «клей».

Клей представляет собой объект, характеризующийся тремя параметрами: величина, растяжимость и сжимаемость. *Величина клея* — то расстояние по горизонтали или вертикали, которое он будет занимать по умолчанию (образно выражаясь, при отсутствии воздействий факторов внешней среды). *Растяжимость* задает то расстояние, которое клей может занять дополнительно, не ухудшая, с точки зрения типографии, качества взаимного расположения боксов. *Сжимаемость* показывает на какое расстояние может быть изменена величина клея в меньшую сторону. Клей никогда не будет сжиматься больше заданной ему сжимаемости. Однако,

если клей имеет положительную компоненту растяжимости, то он может растягиваться бесконечно, однако превышение заданного значение влечет значительные штрафы.

В общем случае параметры клея задаются как

```
<размер1> [plus<размер2>] [minus<размер3>]
```

где `<размер1>` величина клея, `<размер2>` его растяжимость, а `<размер3>` — сжимаемость. Все размеры задаются с обязательным указанием единиц измерения, как было описано ранее. Задание растяжимости и сжимаемости не является обязательным, если они не заданы, то они считаются равными нулю.

Горизонтальный клей задается командой `\hskip`, вертикальный — командой `\vskip`. Существует ряд команд, которые задают клей, уже определенный ранее. Например,

```
\smallskip, \medskip, bigskip.
```

Если при склеивании боксов (горизонтально или вертикально) на одной линии будет установлен клей с разными параметрами сжимаемости и растяжимости, то и сжиматься или растягиваться клей будет пропорционально величине этих параметров. Учитывая это, интересные эффекты возникают когда между боксами размещается клей с бесконечной растяжимостью. Если он расположен в строке, и слева и справа от него есть боксы, то он займет все пространство между ними, раздвинув строку. Если такой клей будет слева, все строка прижмется вправо, и наоборот. Если же два таких клея будут по краям справа и слева в строке, то остальные символы будут расположены точно по центру, именно так и реализована команда центрирования строк.

Горизонтальный клей с бесконечной растяжимостью устанавливается командой `\hfill`, вертикальный командой `\vfill`. Клей, которые имеет как бесконечную растяжимость, так и бесконечную сжимаемость, устанавливается командами `\hss` и `\vss`.

Для работы с клеем используются регистры клея `\skip0`...`\skip255`. Аналогично регистрам размера или числовым регистрам, для них работают команды `\newskip`, `\skipdef` и команды арифметических операций.

Лекция 10

Таблицы в LaTeX

Процедура tabular, оформление элементов таблиц, пакеты longtable и supertabular для верстки длинных таблиц.

1. Строение таблицы

Таблица является наглядным средством структурирования информации и считается одним из самых сложных текстовых элементов. Таблица — это особая форма передачи содержания. Далеко не всякое содержание следует облекать в форму таблицы. Основную сложность составляет не столько оформление, сколько выбор данных для таблицы. Чем богаче связи данных, которые требуется отразить, тем сложнее выглядит таблица.

При грамотном написании «исходника» любые изменения, затрагивающие структуру таблицы, делаются в LaTeX очень быстро.

Содержимое таблицы организуется в колонки (графы) и горизонтальные строки таким образом, что каждый элемент является составной частью и строки, и колонки.

Основные элементы таблицы:

- нумерационный и тематический заголовки (номер таблицы и её название);
- головка (заголовочная часть таблицы);
- хвост (вся остальная часть таблицы без головки);
- боковик (первая слева графа таблицы);
- прографка (хвостовая часть таблицы без боковика).

Все данные таблицы содержатся в прографке.

Характеристики, общие для каждого элемента графы, выносятся в заголовок таблицы, а общие для каждого элемента строки — в заголовок боковика. Вместо отсутствующих данных в таблице принято размещать

многоточие; если же данных в принципе быть не может, то это отмечается с помощью тире. Рамки не являются обязательным элементом таблицы.

Для изображения простейших таблиц достаточно LaTeX'овского окружения `tabbing`, которое воспроизводит принцип печати таблицы на печатной машинки. Этот принцип легко просматривается из примера:

```
\begin{tabbing}
MMMMMMMMMMM \= MMMMMM\= \kill
СИБИРЬ\> Лето \> Зима \\
Жара  \> да   \> нет \\
Мороз \> нет  \> да
\end{tabbing}
```

СИБИРЬ	Лето	Зима
Жара	да	нет
Мороз	нет	да

Первая строка — управляющая. Команда `\=` устанавливает положение табулятора. Команда `\kill` завершает управляющую строку. В результате действия этой команды первая строка не будет напечатана, но все позиции табуляции, установленные в ней, запомнятся LaTeX'ом, и их можно будет использовать в последующих строках.

Остальные строки печатаются, в них переход к следующей табуляции осуществляется с помощью команды `\>`, а перевод строки завершается комбинацией `\\` (роль возврата каретки печатной машинки). Окружение формирует абзац, состоящий из отдельных строк. Поэтому часть таблицы легко переносится на следующую страницу.

Пользуясь окружением `tabbing`, необходимо самостоятельно следить, чтобы разные колонки не накладывались друг на друга.

Для печати более сложных таблиц окружение `tabbing` используются редко, поскольку для этого в LaTeX'е предусмотрены более удобные средства — окружения `tabular, array, longtable, superyabular, avtotab` и другие.

Самым популярным окружением для отображения таблиц в LaTeX является окружение `tabular`. Окружение `array` практически полностью идентично `tabular` по своим возможностям. Отличие этих окружений состоит в том, что `array` работает в математической моде, в лекции 5 мы использовали его для создания матриц, которые по своей сути тоже являются таблицами.

2. Возможности окружения tabular

В общем случае `tabular` определяется следующим образом:

```
\begin{tabular}[позиционирование таблицы]%
                                  {формат столбцов}
                <колонки таблицы>
\end{tabular}
```

Окружению `tabular` необходимо задать обязательный аргумент — преамбулу таблицы.

```
\begin{tabular}{lrc}
<колонки таблицы>
\end{tabular}
```

Преамбула, помещаемая в фигурных скобках непосредственно после `\begin{tabular}`, представляет собой, в простейшем случае, последовательность букв, описывающих структуру колонок таблицы (буква соответствует колонке). Стандартные случаи:

l — колонка выравнивается по левому краю;

r — колонка выравнивается по правому краю;

c — колонка с центрированным текстом.

Между `\begin{tabular}` и закрывающей окружение командой `\end{tabular}` располагается собственно текст таблицы. Данные делятся на ячейки с помощью символа «логическое И» `&`. Переход на следующую строку контролируется стандартной командой переноса строки `\\`. Пробелы в начале или конце «графы» таблицы игнорируются.

Приведенную выше таблицу в окружении `tabular` можно набрать так:

```
\begin{tabular}{lcc}
\ttfamily СИБИРЬ& Лето & Зима \\[.5cm]
\ttfamily Жара  & да   & нет \\
\ttfamily Мороз & нет  & да
\end{tabular}
```

И получится

СИБИРЬ	Лето	Зима
Жара	да	нет
Мороз	нет	да

Заметим, что команда `\\`, завершающая первую строку, дана с необязательным аргументом. Он имеет следующий смысл: после строки вставляется дополнительный вертикальный промежуток. Заметим также, что команда `\ttfamily` всегда меняла шрифт только в одной графе таблицы, не действуя на соседние. Это объясняется тем, что графа таблицы образует группу, так что любые изменения параметров (в том числе текущего шрифта), проведенные в одной графе, не влияют на остальные.

Кроме того, в описании колонок в преамбуле задаётся информация о вертикальных линиях (разделяющих столбцы). По умолчанию, таких линий нет вообще. Если в описании колонок поставить символ `|`, то в соответствующем месте будет одинарная вертикальная линия, если поставить `||` — двойная. Если после команды `\\`, завершающей строку, поставить `\hline`, то будет напечатана горизонтальная линия, разделяющая эту и следующую строки. По аналогии с двойной вертикальной чертой две команды `\hline` формируют двойную горизонтальную линию.

Вся таблица, порождаемая окружением `tabular` (и `array`), рассмат-

ривается TEX'ом как одна большая буква (если окружение `tabular` встретилось в середине абзаца, то эта «буква» будет помещена в строку). По этой причине таблица, созданная в окружении `tabular` не может размещаться сразу на нескольких страницах.

Перечислим опции в LATEX'овских окружениях `tabular` и `array`:

Стандартные опции	
`l`	выровнять колонку по левому краю
`c`	отцентровать колонку
`r`	выровнять колонку по правому краю
`p{ширина}`	абзац(ы) заданной ширины выровнять по верхним строкам
`@{команды}`	вместо межколоночного просвета вставить команды
Дополнительные опции	
`m{ширина}`	В отличие от p{ширина} центрировать по вертикали.
`b{width}`	В отличие от p{ширина} выровнять по нижним строкам.
`>{команды}`	(Перед l, r, c, p, m или b) Вставляет команды непосредственно перед содержимым колонки. В качестве команды можно задать шрифт для всей колонки, ненулевой абзацный отступ (к примеру, `\setlength{\parindent}{1cm}` — значение, добавляемое к высоте каждой клетки таблицы) или открыть формулу (`$`).
`<{команды}`	(После l, r, c, p, m или b) Вставляет команды непосредственно после содержимого колонки, например закрывающий формулы $.
`|`	Вставляет вертикальную черту |, при этом расширяет расстояние между колонками на ширину черты.
`!{декл}`	Вставляет `декл`, не ликвидируя просвета между колонками.
`Φ`	Если иная буква, например, Φ, заранее определена командой `\newcolumntype{Φ}{>{$}c<{\$}}`, то эта буква в преамбуле равнозначна фрагменту `>{$}c<{$}`.

3. Пакеты для оформления элементов таблицы

3.1. Объединение клеток таблицы

Для объединения рядом расположенных по горизонтали клеток можно воспользоваться командой

```
\multicolumn{n}{формат колонки}{текст}.
```

Первый обязательный параметр соответствует числу объединённых колонок, второй — формату получившейся объединённой колонки, третий — текст. Для объединения клеток по вертикали можно воспользоваться пакетом `multirow` в котором определяется набор одноимённых команд:

В этом пакете определены две команды с двумя и тремя обязательными аргументами:

```
\multirow{<число строк>}{<ширина>}{<текст>}
\multirow{<число строк>}*{<текст>}
```

В качестве первого аргумента передаётся число строк, которые займёт объединённая клетка, далее можно выбрать либо автоматическое вычисление ширины, или указать её самостоятельно. Число строк может быть отрицательным. В этом случае объединяются ячейки сверху от команды. Покажем, как в одной и той же таблице можно объединить колонки и столбцы:

```
\begin{tabular}{|*{3}{c|}}
\textbf{ИЛИ} & Истина & Ложь\\[2mm]\hline\hline
Истина & \multicolumn{2}{c|}{Истина}\\
Ложь & Истина & Ложь
\end{tabular}

\begin{tabular}{|*{3}{m{1.2cm}|}}
\textbf{ИЛИ} & Истина & Ложь\\[2mm]\hline\hline
Истина & \multirow{2}{1.2cm}{Истина} & Истина \\
Ложь & & Ложь
\end{tabular}
```

Объединены колонки:

ИЛИ	Истина	Ложь
Истина Ложь	Истина Истина	Ложь

Объединены столбцы:

ИЛИ	Истина	Ложь
Истина Ложь	Истина	Истина Ложь

Если таблица, в которой используется `\multicolumn`, является еще и линованной, то возможностей команды `\hline` для рисования горизонтальных отрезков может не хватить. Действительно, иногда бывает нужен горизонтальный отрезок, простирающийся не на всю ширину таблицы, а охватывающий только часть ее колонок. Для рисования таких отрезков предусмотрена команда `\cline`. Как и `\hline`, ее нужно давать сразу после `\\`. Команда `\cline` имеет один обязательный аргумент — номера первой и последней из колонок, охватываемых горизонтальной чертой, разделенные знаком «-». Способ использования команды `\cline` будут ясен из анализа исходного файла для следующей таблицы:

```
\begin{tabular}{clp{.3\textwidth}}\multicolumn{2}{c}%
{\bfseries Компоненты учебно-методического комплекса}
            &\bfseries назначение\\[\smallskipamount]
преподавателю &  $\left\{\begin{tabular}{l}
        конспект лекций\\методическое руководство\\
                          \end{tabular} \right.$
```

```
&для организации изучения материала\\[\bigskipamount]
студенту        &
$\left\{ \begin{tabular}{l} конспект лекций\\задачник\\
         \end{tabular}\right.$
&для изучения и закрепления материала\\[\bigskipamount]
обоим           &
$\left\{ \begin{tabular}{l} экзаменационные билеты\\
 тесты\\ \end{tabular}\right.$
& для контроля\\[\bigskipamount]       необходимо:
& указания для самостоятельной работы& для изучения\\
желательно:   & презентация лекций   &для наглядности\\
требуется$^1$:& тесты   &  для контроля\\ \cline{1-2}
\multicolumn{3}{l}{$^1$\rule{0pt}{11pt}\footnotesize{
   Используется только для самоконтроля студентов.}}\\
\end{tabular}
```

После компиляции получим такую таблицу:

Компоненты учебно-методического комплекса		**назначение**
преподавателю	{ конспект лекций методическое руководство	для организации изучения материала
студенту	{ конспект лекций задачник	для изучения и закрепления материала
обоим	{ экзаменационные билеты тесты	для контроля
необходимо:	указания для самостоятельной работы	для изучения
желательно:	презентация лекций	для наглядности
требуется[1]:	тесты	для контроля

[1] Используется только для самоконтроля студентов.

Разберем, как устроена эта таблица. Аргумент `{cll}` означает, что она состоит из трех колонок, из которых левая центрирована, а две другие прижаты влево. Три последние графы набраны без ухищрений. Заголовок таблицы сделан с помощью команды `\multicolumn`; команда `\\`, завершающая первую строку таблицы, имеет необязательный аргумент; это сделано, чтобы отодвинуть заголовок по вертикали от остальной части таблицы.

Рассмотрим теперь, как устроена вторая графа (начинающаяся с «преподавателю»). Следующий текст образует в таблице одну «запись» (часть таблицы, расположенную на пересечении графы и колонки):

$$\left\{\begin{array}{l}\text{конспект лекций}\\ \text{методическое руководство}\end{array}\right.$$

Фигурная скобка получена с помощью команд `\left` и `\right`, применяемых при наборе формул. Так как эти команды вне формул использовать нельзя, то этот фрагмент текста оформлен как формула.

Команды `\\`, завершающие первые три графы основной части таблицы, имеют необязательные аргументы, задающие дополнительные вертикальные пробелы после этих граф (иначе фигурные скобки будут упираться друг в друга и портить вид таблицы).

К слову «требуется» в последней строке таблицы дана сноска. Знак сноски тоже реализован как математическая формула — верхний индекс 1 к «пустой формуле»; текст сноски реализован как графа таблицы, охватывающая все три колонки (с помощью команды `\multicolumn`. Команда `\footnotesize` задает размер шрифта, используемый в обычных сносках. Линия, отделяющая сноску от остальной части таблицы, реализована с помощью команды `\cline`. в строке Наконец цифра 1 в самом тексте сноски определена как `$^1$\rule{0pt}{11pt}|`, чтобы команда `\rule` задала невидимый символ, занимающий по вертикали 11 пунктов и не занимающий места по горизонтали. Этот невидимый символ поставлен в качестве подпорки: без нее горизонтальная черта соприкасалась бы с цифрой 1.

3.2. Косая черта в заголовке таблицы

Иногда для оформления клетки на перекрестии боковика и головки уместно использовать косую черту. В пакете `slashbox` определена команда `\backslashbox{текст}{текст}`, делящая клетку на две части наклонной чертой. Это может пригодиться, например, для следующего вида таблиц:

```
\begin{tabular}{|l||*{4}{c|}}\hline
\backslashbox{Помещение}{Дата}
&\makebox[3em]{28/10}&\makebox[3em]{29/10}
&\makebox[3em]{30/10}&\makebox[3em]{31/10}\\ \hline
\hline Малый зал              &&&&\\\hline
       Лекционная аудитория &&&&\\\hline
       Малая аудитория       &&&&\\\hline
\end{tabular}
```

Помещение \ Дата	28/10	29/10	30/10	31/10
Малый зал				
Лекционная аудитория				
Малая аудитория				

Можно добавлять новые колонки (`\\`) в «Помещение» и/или «Дата». Заметим, что иногда пространство перед косой чертой может получаться слишком большим (если колонка достаточно длинная). Управлять длиной черты, разбивающей ячейку, можно при помощи необязательного аргумента `\backslashbox[x]{боковик}{головка}`, как, например, в следующем примере преамбулы таблицы:

```
\begin{tabular}{|l||*{4}{c|}}\hline
\backslashbox[48mm]{Помещение}{Дата}
```

Дата / Помещение	28/10	29/10	30/10	31/10
Аудитория с невероятно длинным названием				
Лекционная аудитория				
Малая аудитория				

Команда `\(back)slashbox` оставляет по умолчанию пустое пространство шириной `\tabcolsep` с обоих сторон от колонки. Регулировать это можно следующим образом:

```
\begin{tabular}%
{|@{\ $\bullet$\hspace*{3mm}}l||*{4}{c|}}
\hline\multicolumn{1}{|@{}l||}%
{\backslashbox[0pt][l]{Помещение}{Дата}}
```

В результате получится такая таблица:

Дата / Помещение	28/10	29/10	30/10	31/10
• Малый зал				
• Лекционная аудитория				
• Малая аудитория				

Здесь `[l]` запрещает дополнительное пространство слева от этой колонки. Аналогично, можно использовать `[r]` и `[lr]`. В этом случае можно было тоже установить ширину ячейки, но мы положили ее равной 0pt.

3.3. Оформление разделительных линий

Возможностей окружений `array` и `tabular` вполне хватает для печати простейших линованных таблиц, но в более сложных случаях возникают проблемы.

Подключение стилевого пакета `hhline` из коллекции `tools` облегчает работу с линованными таблицами. В этом пакете доступны уже рассматри-

вавшиеся команды `\hline` и `\cline`. Кроме того, стиль `hhline` определяет команду `\hhline`, которая позволяет управлять созданием двойной рамки, не создавая ненужных пересечений.

Отметим особо, что команда `\hline` обрабатывает пересечения линеек независимо от того, какие вертикальные линейки заданы в преамбуле. Поэтому пользователь сам должен заботиться о том, чтобы аргумент `\hline` был согласован с преамбулой.

В аргументе команды `\hline` в фигурных скобках описывается как вид линейки, так и вид ее пересечения с вертикальными линейками. Вот первый пример использования `\hline`:

зима	снег	гололед	холод
лето	солнце	засуха	жара

```
\begin{tabular}{|c|cc|c|}
\hline зима & снег & гололед & холод\\
\hhline{|=|~~|-|}
лето & солнце & засуха & жара\\ \hline
\end{tabular}
```

Аргумент команды `\hline` устроен следующим образом. Во-первых, в нем сказано, что на территории первой колонки линейка должна быть двойной (символ «=»), на территории второй и третьей колонок линейки не должно быть вовсе (символ ~ — «тильда»), а на территории четвертой колонки линейка должна быть одинарной (символ «-»). Если в таблице n колонок, то в аргументе `\hhline` должны присутствовать n символов «–», «=» или «~», имеющих тот же смысл, что и выше.

Между символами, описывающими поведение линейки внутри колонок, расположены символы, описывающие пересечения горизонтальной линейки с вертикальными. В нашем примере это были вертикальные черточки «|»; кроме них, для задания информации о пересечениях линеек можно использовать символы «:», «#», а также буквы «t» и «b». Примеры значений

аргумента команды `\hhline`:

|t: — уголок: пересечение вертикальной и горизонтальной двойной линии, расположенный слева наверху;

:t| — уголок: пересечение вертикальной и горизонтальной двойной линии, расположенный справа наверху;

|b: — уголок: пересечение вертикальной и горизонтальной двойной линии, расположенный слева внизу;

:b| — уголок: пересечение вертикальной и горизонтальной двойной линии, расположенный справа внизу;

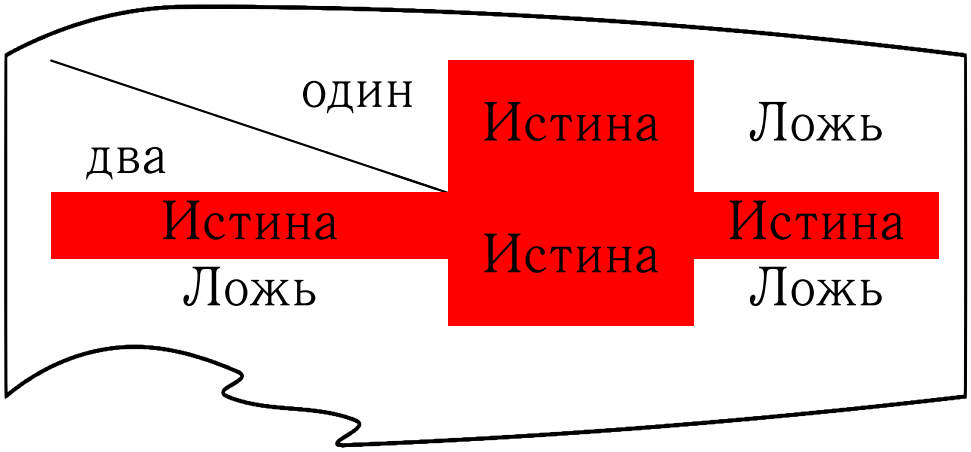
|: — пересечение вертикальной и горизонтальной двойной линии, расположенной слева, но не в уголке;

:| — пересечение вертикальной и горизонтальной двойной линии, расположенной справа, но не в уголке.

В случае использования пунктирных разделительных линий необходимо подключать пакет `arydshln`. Хорошая документация к этом пакету содержится в файле `arydshln-man.pdf`.

3.4. Раскрашивание таблицы

Для раскрашивание таблицы в разные цвета предназначен пакет `colortbl` (см. документацию `colortbl.pdf`). Покажем возможности совместного употребления пакетов `colortbl` и `multirow`. Таблица:



набрана следующим образом:

```
\begin{tabular}%
{c>{\columncolor{red}}cc}
\backslashbox{два}{один}&Истина&Ложь\\
\rowcolor{red}
Истина & & Истина \\
Ложь & \multirow{-2}*{Истина} & Ложь
\end{tabular}
```

3.5. Автоматическая нумерация строк таблицы

Окружение `listliketab` из одноимённого пакета специализируется на создании таблиц, оформленных как перечисление. Это окружение очень подходит для оформления вопросников:

1. **Первый** или **второй** или **третий**
2. **Пар** или **вода** или **лед**
3. **Красный** или **желтый** или **зеленый**

```
\storestyleof{enumerate}   \begin{listliketab}
\newcounter{tabenum}\setcounter{tabenum}{0}
\newcommand{\nextnum}%
{\addtocounter{tabenum}{1}\thetabenum .}
\begin{tabular}{L>{\bfseries }l
@{ или }>{\bfseries }l@{ или }>{\bfseries } l }
\nextnum & Первый & второй & третий \\
\nextnum & Пар  & вода & лед \\
\nextnum & Красный & желтый & зеленый \\
\end{tabular}                  \end{listliketab}
```

4. Таблицы, занимающие несколько страниц

Как уже отмечалось, окружения array и tabular рассматриваются LaTeX'ом как одна большая буква, и потому не разбивается по страницам. Можно, однако, создавать таблицы, в которых и разбиение на страницы, и определение ширины колонок происходит автоматически.

Для вёрстки длинных (занимающих несколько страниц) таблиц обычно используют одно из двух окружений: `supertabular` или `longtable` определённых в одноимённых пакетах. Эти окружения обладают схожей функциональностью, но разными способами оформления. Оба предоставляют возможность создания стандартной головки и стандартного окончания таблицы, появляющихся в начале, на каждой новой странице и в конце таблицы.

Пакет `longtable`, в отличии от `supertabular`, гарантирует неизменность ширины столбцов на протяжении всей таблицы, что достигается за несколько проходов LaTeX. Неизменность ширины столбцов в зависимости от ситуации может рассматриваться как преимущество, так и как недостаток. В пакете `supertabular` кроме окружения `supertabular` определено окружение `mpsupertabular`, которое каждый отдельный кусок таблицы на своей странице заключает в окружение `minipage`. Это позволяет использовать подстрочные примечания прямо в таблице.

Опишем теперь некоторые особенности окружения longtable, связанные с разбиением на страницы.

Как и окружение tabular, оно принимает один обязательный параметр — преамбулу (устроенную точно так же, как у tabular); внутри окружения действуют в точности те же правила записи текста, что и в окружении tabular (в частности, допустимы команды `\hline`, `\cline` и `\multicolumn`). Разница с окружениями array и tabular состоит в том, что получаемая при этом таблица может занимать несколько страниц и иметь произвольную длину.

После первого запуска LaTeX'а колонки таблицы, определенной как longtable, могут оказаться невыровненными (это связано с тем, что при первом проходе LaTeX читает такую таблицу не целиком, а по кускам, и

выравнивает эти куски независимо друг от друга). Чтобы добиться выравнивания, надо еще раз запустить LaTeX.

Рассмотрим, как реализуется возможность повторения заголовка таблицы на каждой новой странице, на которой таблица продолжается. Для этого надо оформить заголовок в виде строки или группы строк (и/или команд `\hline`), и при этом последнюю из этих строк надо завершить не командой `\\`, а командой `\endhead`.

Часто повторяющийся заголовок не идентичен тому, который ставят в самом начале таблицы. Отдельный заголовок для начала таблицы также оформляют в виде одной или нескольких строк, последняя из которых завершается командой `\endfirsthead`. Кроме того, можно предусмотреть специальную группу строк, которая будет ставиться на каждой странице внизу таблицы. Для этого надо записать строку или группу строк, завершив последнюю из них не командой `\\`, а командой `\endfoot`.

Можно также предусмотреть отдельную группу строк, которая ставится внизу таблицы только на последней из занимаемых таблицей страниц. Для этого надо вместо `\endfoot` написать `\endlastfoot`. Группы строк, завершающиеся командами `\endhead`, `\endfirsthead`, `\endfoot` или `\endlastfoot`, должны стоять в начале окружения longtable. Вот пример использования этих средств:

Итоги аттестации студентов

Ступень	Фамилия	Имя	Оценка
1	2	3	4
бакалавр	Иванов	Иван	отл.
бакалавр	Петров	Петр	хор.
бакалавр	Сидоров	Роман	отл.
бакалавр	Иванова	Анна	хор.
магистрант	Петрова	Мария	отл.

Продолжение на следующей странице

1	2	3	4
магистрант	Сидорова	Елена	хор.
магистрант	Савельева	Елена	отл.
магистрант	Медведева	Ирина	отл.

```
\begin{longtable}{||l|l|l|l||}
\multicolumn{4}{c}{Итоги аттестации %
студентов}\\ \hhline{|:====:|}
Ступень & Фамилия & Имя & Оценка\\
\hhline{|:====:|}1&2&3&4\\
\hline            \endfirsthead  \hline
1&2&3&4\\ \hline  \endhead       \hline
\multicolumn{4}{c}%
{\textit{Продолжение на следующей странице}}
\endfoot \hline\endlastfoot
бакалавр & Иванов & Иван & отл.\\
                    <...>
магистрант & Савельева & Елена & отл.\\
\end{longtable}
```

В таблице, оформленной с помощью longtable, можно явно указать место разрыва страницы с помощью `\newpage` (а также `\pagebreak` в окружении longtable или `\nopagebreak`). Эти команды должны следовать непосредственно после `\\` (можно с новой строки).

При пользовании пакетом longtable предусмотрена возможность автоматической нумерации таблиц, созданных окружением longtable. Для этого используется та же команда `\caption`, в окружении longtable что и в окружениях table или picture, но пользоваться ей надо чуть по-другому. А именно, после этой команды необходимо поставить `\\`, `\endhead` или `\endfirsthead` (или `\endfoot`,`\ldots`).

При этом автоматически нумеруемые таблицы можно обычным образом метить с помощью `\label` (и потом ссылаться на эти места с помощью

`\ref`), но необходимо соблюдать два ограничения: метку нельзя ставить в строки, которые появятся на печати более одного раза (благодаря повторению заголовков на других страницах); с команды `\label` не должна начинаться ни одна графа таблицы. На автоматически нумерующийся заголовок выделяется по умолчанию 4 дюйма. Чтобы изменить этот размер, надо присвоить соответствующее значение параметру `\LTcapwidth`.

В отличие от окружения tabular, таблица в окружении longtable не рассматривается как одна большая буква, а сразу располагается между абзацами, по умолчанию — по центру. Чтобы таблица была прижата к правому краю, надо указать у окружения longtable необязательный аргумент r (в квадратных скобках, между `\begin{longtable}` и преамбулой); необязательный аргумент l даст таблицу, прижатую влево.

5. Импорт содержимого таблицы из базы данных

Подобно тому, как BibTEX генерирует список литературы из библиографических баз данных, окружение `autotab` (пакет был разработан Gabriele Kruljac) позволяет генерировать строки таблицы из файла данных с расширением `.dat`.

Заметим, что данные должны вводиться с правильным TEX'овским синтаксисом.

Имя вводимого файла данных считывается каждый раз, когда в основном документе встречается команда `\readtabline{...}`. Файл `.tex` обрабатывается через командную строку любого текстового редактора (например, `Far`).

Вид «минимального» исходного файла:

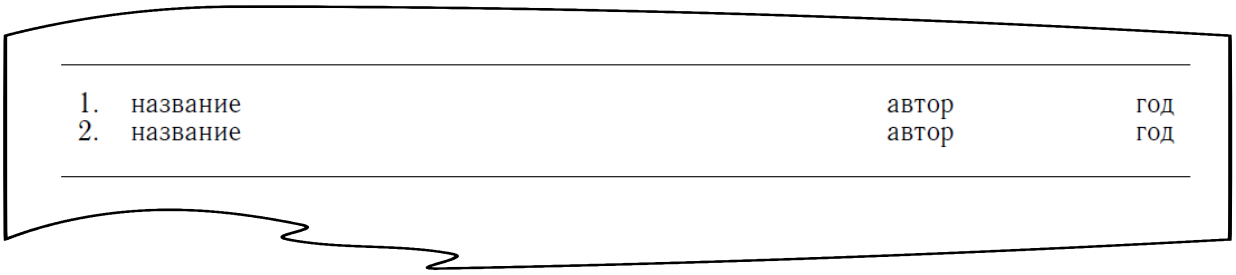
```
\documentclass[11pt]{article}
\usepackage[russian]{babel}
\usepackage{amssymb}%необязательно
\usepackage{autotab}
\begin{document}
            <<...>>
```

```
            <<...>>
\begin{tabular}{lp{10cm}p{3cm}l}
\hline\\
\noalign{\readtabline{4}}
\autotabline\\
\hline
\end{tabular}
            <<...>>
\ end{document}
```

Файл данных — это обычный текстовый файл, записанный с расширением `.dat`. Содержимое каждой ячейки таблицы заносится с новой строки. Пример содержимого файла данных `имяфайла.dat`:

```
1.
название
автор
год
2.
название
автор
год
```

А так выглядит результат:



1.	название	автор	год
2.	название	автор	год

Дополнительное подключение пакета `supertabular` позволяет автоматически верстать сколь угодно длинные таблицы со сложными заголовками. Пакет `autotab` очень удобен для таблиц, содержимое которых постепенно построчно добавляется, например, для изготовления списка научных и методических публикаций.

Лекция 11

Управление формой абзацев, бегущие заголовки и плавающие вставки

Возможности LaTeX'а по управлению форматированием абзацев, бегущих заголовков и плавающих вставок.

1. Управление формой абзацев

1.1. Управление способом обрывания строки

Режим форматирования абзацев LaTeX позволяет повлиять на режим форматирования целых абзацев. Декларации

```
\sloppy    \fussy
```

изменяют способ обрезания строки. Декларация `\fussy`, действующая по умолчанию, препятствует образованию слишком больших пробелов между словами, но разрешает словам вылезать за правую границу страницы, если не найдена подходящая точка для переноса. Декларация `\sloppy` почти всегда обрывает строку в точности на правом краю, но при этом могут образоваться очень длинные промежутки между словами. В этом случае LaTeX выдаст предупреждение `Underfull \hbox` о незаполненном горизонтальном боксе. Область действия декларации `\sloppy` определяется следующим образом. LaTeX выполняет разбиение на строки, когда доходит до конца абзаца (то есть до пустой строки) или до процедуры, начинающей печать с новой строки. Следовательно, форматирование абзаца осуществляется в соответствии с декларацией, действующей на этот момент. В остальном область действия декларации `\sloppy` определяется как обычно. Она простирается до декларации `\fussy`, возвращающей LaTeX в обычный режим, или же может быть ограничена фигурными скобками.

Как будет выглядеть при этом абзац, заранее предсказать трудно. Какие-то абзацы могут смотреться приемлемо, какие-то — чудовищно. Поэтому ставить `\sloppy` в преамбулу, задавая тем самым этот режим на

весь текст, несколько рискованно. Разумнее задать эту команду перед концом того абзаца, в котором произошел `overfull`, и посмотреть, что из этого получится.

Вот пример случая, в котором может потребоваться управление обрыванием строки:

`Математические символы подразделяются на несколько классов, различающихся, главным образом, по способу позиционирования символов относительно соседних.`

Применение команд `\sloppy` и `\fussy` дают разные результаты:

Математические символы подразделяются на несколько классов, различающихся, главным образом, по способу позиционирования символов относительно соседних.

Математические символы подразделяются на несколько классов, различающихся, главным образом, по способу позиционирования символов относительно соседних.

После последних строк левого и правого абзацев, были даны команды, соответственно:

```
... символов относительно соседних.
{\sloppy


}
```

```
... символов относительно соседних.
{\fussy

}
```

Во втором случае почти в каждой строке будет переполнение, а в первом случае — разряженность.

1.2. Абзацы нестандартной формы

Абзацы с регулируемыми отступами

Пусть нам потребовалось создать абзац с «отрицательным» абзацным отступом, в котором все строки, кроме первой, начинаются на расстоянии $2\,cm$ от полей. Этого можно добиться следующим образом:

```
\hangindent=2cm \noindent
Здесь \TeX'овский парамет ...
```

Здесь TEX'овский параметр `\hangindent` (по-английски: hanging indentation) означает величину отступа от полей во всех строках абзаца, кроме первой (по умолчанию значение этого параметра равно нулю). Мы начали абзац командой `\noindent`, чтобы первая строка не началась с абзацным отступом.

Пусть теперь нам хочется, чтобы дополнительный отступ, величина которого задана параметром `\hangindent`, начинался не со второй строки, а, скажем, с третьей. Для этого надо установить еще один TEX'овский параметр, обозначаемый `\hangafter`.

Команды `\hangindent=3cm \hangafter=2 \noindent`, поставленные в начале параграфа, который вы читаете, позволяют сделать так, чтобы отступ начался не с первой строки, а там, где нам это потребовалось — со второй. Значение параметра `\hangafter` — номер строки, после которой начинается дополнительный отступ. По умолчанию значение `\hangafter` равно единице (как и было в нашем первом примере).

Можно также добиться того, чтоб дополнительный отступ не начинался после какой-то строки, а напротив, присутствовал только в нескольких первых строках абзаца.

Для этого надо присвоить параметру `\hangafter` отрицательное значение: если величина `\hangafter` равна $n < 0$, то дополнительный отступ, равный `\hangindent`, будет присутствовать в строках номер $1, 2, \ldots |n|$. С помощью рассмотренных средств TEX'а можно выкапывать в абзацах небольшие ямки. На место образующегося белого прямоугольника можно, например, вклеить иллюстрацию. Данный параграф начинается с команд

`\hangindent=5cm\hangafter=-3 \noindent`.

Можно сделать так, что дополнительный отступ, задаваемый параметром `\hangindent`, будет делаться не слева, а справа.

Для этого значение этого параметра надо сделать отрицательным. Именно, если выбранное нами значение `\hangindent` равно $h < 0$, то дополнительный отступ размером $|h|$ будет отсчитываться от правого, а не левого поля (в каких именно строках будет этот дополнительный отступ, по-прежнему определяется значением `\hangafter`). Данный параграф начинается с команд `\hangindent=-5cm\hangafter=-3 \noindent`.

Начало исходного текста для следующего причудливого абзаца выглядело так:

```
\parshape=14
0cm 6cm  .1cm 5.8cm  .17cm 5.66cm  .5cm 5cm
.9cm 4.2cm  1.05cm 3.9cm  1.1cm 3.8cm  1.1cm 3.8cm
1.05cm 3.9cm  .9cm 4.2cm  .5cm 5cm .17cm 5.66cm
.1cm 5.8cm  0cm 6cm
\noindent \small
Если вам не хватает возможностей...
```

А результат выглядит так:

Если вам не хватает возможностей, приведенных выше, то посмотрите, как можно с помощью TEXá создать абзац совсем уж причудливой формы. Все переносы в словах и места разрывов строк были найдены TEX'ом автоматически. Получившийся абзац напоминает автору — С.М. Львовскому изображение шахматной ладьи. (Предложение в скобках пришлось дописать, чтобы изображение не выглядело незавершенным.)

Смысл текста в начале, благодаря которому абзац выглядит таким красивым, такой: число 14, следующее непосредственно после `\parshape` и знака равенства, задает количество строк, имеющих нестандартные длину и/или отступ от левого поля. После этого числа, через пробел (конец строки, как мы помним, — тоже пробел), перечислены отступы от левого поля и длины строк: `0cm` — отступ первой строки от левого поля, `6cm` — ее длина, `.1cm` — отступ второй строки от левого поля, `5.7cm` — ее длина, и т. д.

Если написано, что `\parshape` равно n, то после этого должно следовать $2n$ длин. Если реально в абзаце получится менее n строк, то указания на длину и отступ отсутствующих строк будут проигнорированы TEX'ом; если строк получается больше, чем n, то все последующие строки будут иметь те же отступ и длину, что заданы для строки номер n.

Абзац начинается командой `\noindent`, чтобы отступ самой первой строки был действительно равен нулю.

После пустой строки или команды `\par` действие параметров, заданных командой `\parshape`, прекращается.

У абзаца, форма которого задана с помощью `\hangindent` или `\parshape`, длина и отступ строки зависят, как вы могли заметить, от ее номера. Если такой абзац содержит выключную формулу, то TEX считает, что эта формула занимает три строки, причем сама формула расположена в средней из этих трех (реально формула может, разумеется, занимать больше места).

Возможности пакета `shapepar`

Для того, чтобы верстать абзацы причудливых форм в стандартную комплектацию LATEX 2_ε входит специальный пакет `shapepar`. Команды пакета придают любому абзацу экзотическую форму из заданного набора. Например, такую форму, как у этого абзаца, можно придать любому другому при помощи команды `\heartpar{текст абзаца}`. О других командах и возможностях пакета можно узнать из спецификации к нему.

♡

Управление формой абзаца вручную

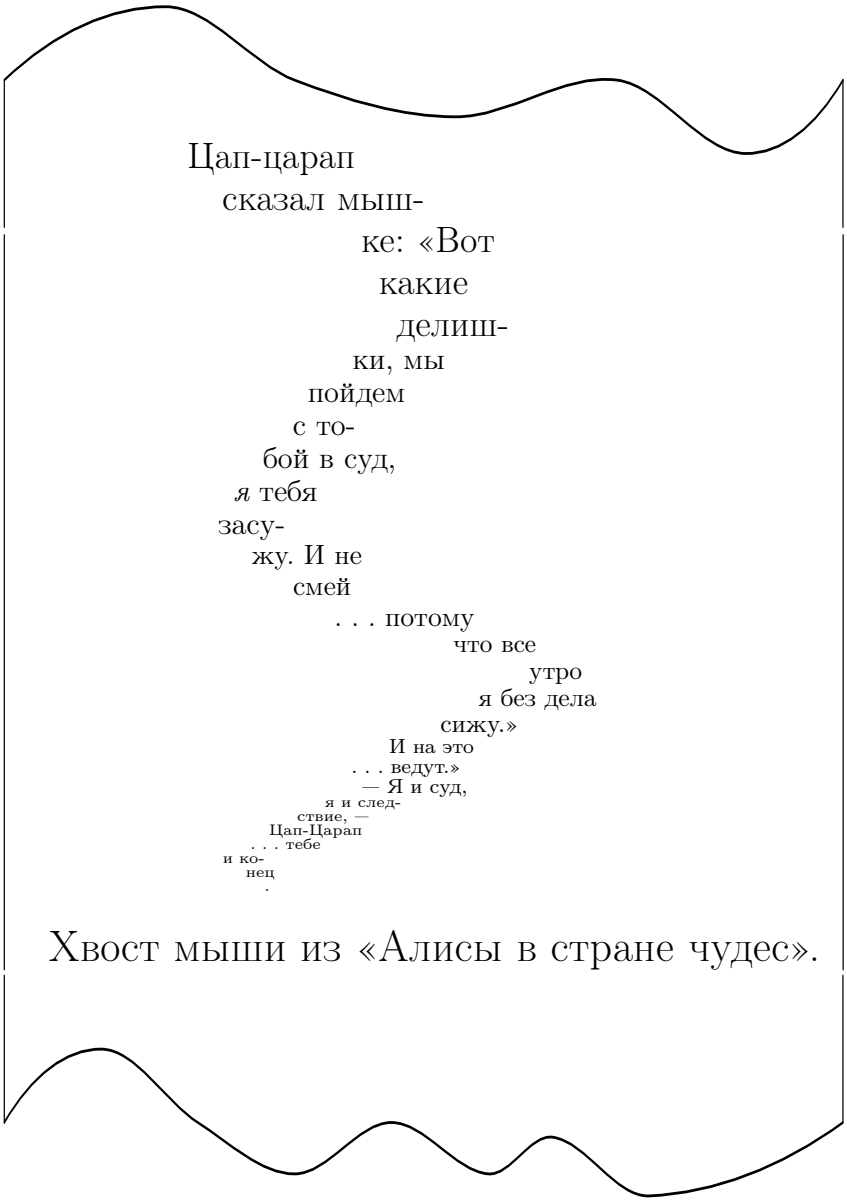
Нестандартный абзац практически любой формы можно изготовить «вручную» при помощи команд `\parindent`, `\setlength` и `\addtolength`.

Команда `\parindent` задает параметр жесткой длины, равный ширине отступа в начале обычного абзаца. В боксе, полученном командой `\parbox`, он равен 0 дюймов. В любом месте значение можно изменить, например, командой `\setlength`.

Команда `\setlength{cmd}{len}` присваивает команде длины `cmd`, которая должна начинаться с бэкслэша, значение `len`. Например, после команды `\setlength{\topmargin}{2in}` параметр `\topmargin` будет равен двум дюймам.

Команда `\addtolength{cmd}{len}` присваивает команде длины `cmd`, которая должна начинаться с бэкслэша, результат сложения `len` с ее текущим значением.

В качестве примера, С.В. Клименко И М.В. Лисина показали, как применить эти команды для изготовления абзаца в форме хвоста мыши из известной книжки «Алиса в стране чудес»:



Хвост мыши из «Алисы в стране чудес».

Идея создания хвоста ясна из фрагмента исходного текста:

```
{\parskip=-1.5pt
{\large \setlength{\parindent}{1in}
Цап-царап   \addtolength{\parindent}{1em}\par
сказал  мыш-\addtolength{\parindent}{4em}\par
ке: <<Вот   \addtolength{\parindent}{.5em}\par
какие       \addtolength{\parindent}{0.5em}\par
делиш-\par}{\normalsize \setlength{\parindent}{1.9in}
ки, мы      \addtolength{\parindent}{-1.5em}\par
пойдем      \addtolength{\parindent}{-.5em}\par
с то-       \addtolength{\parindent}{-1em}\par
бой в суд,  \addtolength{\parindent}{-1em}\par
{\it я} тебя\addtolength{\parindent}{-.5em}\par
засу-\par}{\small \setlength{\parindent}{1.35in}
жу. И не    \addtolength{\parindent}{1.5em}\par
    . . .
и ко-       \addtolength{\parindent}{1.25em}\par
нец         \addtolength{\parindent}{1em}\par
.~          \addtolength{\parindent}{.75em}\par~\par}}
```

1.3. Расширенные возможности форматирования абзацев

Возможности простейших процедур форматирования абзацев (таких как `enumerate` и многих других) расширяются при загрузке дополнительных пакетов.

Пакет `enumerate`

Пакет `enumerate` из коллекции `tools` переопределяет одноименную процедуру, упрощая способ изменения формата меток записей, который теперь можно указать в опции процедуры:

```
\begin{enumerate}[marks] item-список \end{enumerate}
```

Опция `marks` специфицирует формат меток записей. Она может содержать один из символов A, a, I, i или 1, которые отвечают представ-

лению счётчика `ctr`, нумерующего записи, соответственно при помощи команд `\Alph{ctr}`, `\alph{ctr}`, `\Roman{ctr}`, `\roman{ctr}` или `\arabic{ctr}`. Более того, `marks` может содержать любые другие символы или команды LaTeX'а, допустимые в необязательном аргументе команды `\item`; однако символы A, a, I, i и 1 должны быть окружены фигурными скобками, если их не следует интерпретировать как представление счётчика. Следующий пример показывает, как оформить процедуру `enumerate`:

```
\begin{enumerate}[{A}I.]
\item Первая запись первого уровня.
\item Вторая запись первого уровня.
\begin{enumerate}[Абзац a)]
\item Первая запись второго уровня.
\item Вторая запись второго уровня.
\end{enumerate}
\begin{enumerate}[Абзац a]
\renewcommand{\alph}[1]{\asbuk{#1}}
\item Первая запись второго списка.
\item Вторая запись второго списка.
\end{enumerate}
\end{enumerate}
```

AI. Первая запись первого уровня.

AII. Вторая запись первого уровня.

Абзац a) Первая запись второго уровня.

Абзац b) Вторая запись второго уровня.

Абзац а Первая запись второго списка.

Абзац б Вторая запись второго списка.

Пакет alltt вводит одноименную процедуру:

```
\begin{alltt} . . . \end{alltt}
```

Она очень похожа на процедуру `verbatim`, но в ней `\, {, }` сохраняют свое обычное значение. Таким образом, другие команды и процедуры могут появляться внутри процедуры `alltt`. В результате оказывается возможным изменить шрифт внутри процедуры с помощью, например, команды `\emph`, ввести файл при помощи команды `\input` или вставить математическую формулу.

```
\begin{alltt}
В процедуре \texttt{alltt} можно
переключать шрифты и набирать
математические формулы. Например:
       \( x\sp{2}+y\sp{2}=Z\sb{0}. \)
\end{alltt}
Пробелы учитываются!
```

В процедуре alltt можно
переключать шрифты и набирать
математические формулы. Например:

$$x^2 + y^2 = Z_0.$$

Пробелы учитываются!

Команды `\(, \sp, \sb и \)`, использованные здесь, чтобы набрать формулу $x^2 + y^2 = Z_0$. Отметим только, что вне процедуры `alltt` ту же формулу можно было бы записать: `$x^{2}+y^{2}=Z_{0}$`, однако внутри `alltt` символы `$, ^, _` не имеют командного значения, которым они обладают в математических формулах.

Пакет verbatim

Этот пакет тоже из коллекции tools, он изменяет одноименную процедуру так, чтобы избавиться от двух ограничений в её применении. Метод обращения к процедуре прежний:

```
\begin{verbatim} . . . \end{verbatim}
\begin{verbatim*} . . . \end{verbatim*}
```

Во-первых, теперь допускаются пробелы между \end и {verbatim}. Во-вторых, процедура может печатать фрагменты текста неограниченных размеров. Платой за новый метод стало одно следствие, заставившее разработчиков LaTeX 2_ε' отказаться от полной замены старого метода новым. Теперь текст, следующий за \end{verbatim} в той же строке, будет полностью проигнорирован.

Эта проблема легко решается: во входном файле этот текст нужно перенести на следующую строку. В дополнение к verbatim пакет вводит процедуру comment, которая пропускает любой текст между \begin{comment} и \end{comment}:

```
\begin{comment} . . . \end{comment}
```

Кроме того, он определяет команды

```
\verbatiminput{file}
\verbatiminput*{file}
```

которые считывают файл file и печатают его содержимое так, как это делают соответственно процедуры verbatim и verbatim*.

2. Размещение плавающих объектов

Большинство публикаций в наши дни содержат множество иллюстраций и таблиц. Эти элементы нуждаются в специальном обращении с ними, так как они не могут быть разбиты между страницами. Одним из выходов было бы начинать новую страницу каждый раз, когда встречается иллюстрация или таблица, слишком большая, чтобы поместиться на текущей странице. Этот подход привел бы к тому, что страницы оставались бы частично пустыми, что смотрится очень плохо.

Для решения этой проблемы любая иллюстрация или таблица, не умещающаяся на текущей странице, может ‘плавать’, перемещаясь на следующую страницу в процессе заполнения текстом текущей. LaTeX предлагает для плавающих объектов два окружения, одно для таблиц и одно для иллюстраций. Чтобы полностью использовать их преимущества, важно примерно представлять, как LaTeX обрабатывает плавающие объекты. Иначе они могут стать источником разочарования из-за того, что LaTeX помещает их не туда, куда вы хотите.

Вначале рассмотрим команды, предоставляемые LaTeX для плавающих объектов. Любой материал, включенный в окружения `figure` или `table`, трактуется как плавающий.

```
\begin{figure}[спецификация размещения]   или
\begin{table}[спецификация размещения]
```

Оба окружения имеют необязательный параметр, называемый спецификацией размещения. Этот параметр используется для указания LaTeX, куда можно перемещать плавающий объект. Спецификация размещения конструируется путем собирания в строчку ключей размещения плавающего объекта.

Ключи h, t, b, p, и ! разрешают помещать объект следующим образом:

h — здесь же, в том самом месте текста, где он появился. Обычно используется для маленьких объектов.

t — наверху страницы

b — внизу страницы

p — на специальной странице, содержащей только плавающие объекты.

! — не рассматривать большинство внутренних параметров, которые могут предотвратить размещение этого объекта.

Например, таблицу можно начать следующей строкой:

```
\begin{table}[!hbp]
```

Спецификация размещения `[!hbp]` позволяет LaTeX разместить таблицу прямо по месту (`h`), или внизу той же страницы (`b`), или на выделенной странице (`p`), независимо от того, насколько хорошо это смотрится. Если никакой спецификации размещения не задано, стандартные классы предполагают `[tbp]`.

LaTeX размещает каждый встреченный плавающий объект в соответствии с заданной автором спецификацией. Если объект нельзя поместить на текущей странице, он откладывается, помещаясь в очередь иллюстраций или в очередь таблиц. Когда начинается новая страница, LaTeX проверяет, можно ли заполнить специальную страницу плавающими объектами из очередей. Если нет, то первый объект из каждой очереди считается только что встретившимся в тексте: LaTeX снова пытается разместить их в соответствии с их спецификациями (за исключением `h`, что уже невозможно). Новые встреченные в тексте плавающие объекты помещаются в соответствующие очереди. LaTeX сохраняет порядок, в котором встретились плавающие объекты соответствующего типа. Поэтому иллюстрация, которую не удается разместить, отталкивает все дальнейшие иллюстрации к концу документа.

Если LaTeX не размещает плавающие объекты так, как планируется, то часто это означает, что только один объект устроил затор в одной из очередей.

Сделаем еще несколько замечания про окружения `table` и `figure`. Командой \caption{текст заголовка} можно задать заголовок для объекта. Увеличивающийся номер и строка «Рисунок» или «Таблица» добавляются LaTeX'ом.

Две команды \listoffigures и \listoftables работают аналогично команде `\tableofcontents`, печатая список иллюстраций или таблиц, соответственно. В этих списках заголовки повторяются целиком. Если используются длинные заголовки, то необходимо предоставить их краткий вариант для включения в списки. Это делается помещением краткого варианта в квадратные скобки после команды `\caption`.

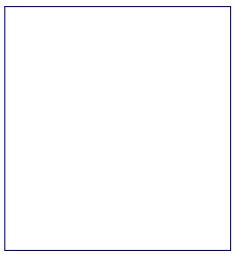
```
\caption[Короткий]{Дддддлллллиииинннннныыыыыыыыыыыыыыйййй}
```

При помощи `\label` и `\ref` можно делать ссылки из вашего текста на плавающий объект.

Следующий пример рисует квадрат и вставляет его в документ. Подобную технику можно использовать, чтобы оставить в документе место под изображения, которые вы вставите позже.

Изображение квадрата, помеченное номером 11, — это место для нового рисунка.

```
\begin{figure}[!hbp]
\makebox[\textwidth]{\framebox[3cm]{\rule{0pt}{3cm}}}
\caption{Три на три сантиметра} \label{white}
\end{figure}
```



Три на три сантиметра

В этом примере LaTeX будет стремиться разместить иллюстрацию прямо по месту (`h`), а случае неуспеха попытается разместить ее внизу страницы (`b`). Если не удастся поместить иллюстрацию на текущей странице, он выяснит, можно ли создать страницу плавающих объектов, содержащую эту иллюстрацию и, возможно, некоторые таблицы из очереди таблиц. Если для отдельной страницы материала еще не накопилось, `h` начинает новую страницу и снова рассматривает иллюстрацию, как если бы она только что появилась в тексте.

В определенных случаях может быть необходимо использовать команду

```
\clearpage или даже \cleardoublepage
```

Она указывает LaTeX'у немедленно разместить все плавающие объекты, остававшиеся в очередях, и затем начать новую страницу. Кроме того, `\cleardoublepage` начинает новую левостороннюю страницу.

Лекция 12 ⏮

Управление компоновкой страницы. Стиль страницы

Настройки стиля страницы, настройка содержимого колонтитулов. Печать в две колонки, пакет multicol.

В этой лекции речь пойдет о том, что определяет стиль страницы и что называют макетом полосы набора.

1. Полоса набора

Полоса набора или страница в печатном документе состоит из трёх частей: верхнего колонтитула, тела страницы и нижнего колонтитула. В тело страницы входит всё, что находится между колонтитулами: текст, подстрочные примечания, рисунки и таблицы. Класс печатного документа определяет размеры всех частей страницы, а также содержание колонтитулов. Левые и правые страницы могут иметь разные размеры и разные колонтитулы, если документ подготовлен для печати на двух сторонах листа бумаги. При двусторонней печати чётные страницы располагаются на левой половине разворота книги, а нечетные на правой.

При односторонней печати все страницы считаются правыми. По умолчанию для печатного документа класса `article, proc, report, slides` и `letter` устанавливается односторонняя печать, для класса `book` двусторонняя.

Опция `twoside` в декларации `\documentclass` поможет установить режим двусторонней печати также для статей и отчетов, но в письмах, слайдах и научных докладах всегда используется односторонний формат страницы. Информация в колонтитулах, которая чаще всего включает в себя номера страниц, а иногда и названия текущей главы и раздела, призвана помочь читателю найти нужное место в печатном документе.

Стиль страницы (полосы набора), установленный классом печатного документа, изменяют декларации

```
\pagestyle{page-style}     \thispagestyle{page-style}
```

Область действия декларации `\pagestyle` подчиняется обычным правилам с одним уточнением: она начинает действовать с текущей страницы. Декларация `\thispagestyle` аналогична `\pagestyle`, но воздействует только на текущую страницу. Аргумент `page-style` может принимать четыре значения:

- `plain` печатает номера страниц в нижнем колонтитуле, а верхний колонтитул пуст. Стандартные классы, кроме `book`, используют стиль `plain` по умолчанию; empty означает, что верхний и нижний колонтитулы пусты. LaTeX поддерживает нумерацию страниц, но номера не печатает;

- `headings` печатает номера страниц и другую информацию, определяемую классом документа (обычно заголовки разделов), в верхнем колонтитуле. Нижний колонтитул пуст. Печатный документ класса `book` использует стиль `headings` по умолчанию;

- `myheadings` означает, что содержание верхнего колонтитула должен задать сам пользователь, используя команды `\markboth` и `\markright`, описанные ниже. Нижний колонтитул пуст.

Декларацию `\pagestyle` обычно вставляют после команд типа `\chapter`, которые начинают печатать текст с новой страницы. Декларации

```
\markboth{left}{right}   и   \markright{right}
```

имеют смысл при выборе стиля страницы `headings` или `myheadings`. В первом случае (`headings`) они переопределяют содержание верхнего колонтитула, установленное классом печатного документа (посредством этих же деклараций). Во втором случае (`myheadings`) использование `\markboth` или `\markright` обязательно. Аргументы `left` и `right` должны содержать ту информацию, которая пойдёт соответственно на левую и правую страницы, причём при односторонней печати все страницы считаются правыми.

В верхний колонтитул левой страницы переносится аргумент `left` последней команды `\markboth` перед концом страницы.

В верхний колонтитул правой страницы переносится аргумент `right` первой декларации `\markright` или `\markboth` на странице, а если их нет, то последней из этих деклараций перед началом страницы.

Как `left`, так и `right` обрабатываются в строковой моде. Они являются подвижными аргументами, поэтому хрупкие команды в них должны быть защищены командой `\protect`.

При выборе стиля страницы `headings` класс печатного документа устанавливает, какие команды секционирования и при помощи какой из команд `\markboth`, `\markright` заносят название раздела в колонтитулы.

Чтобы переопределить эти установки, необходимо поместить `\markboth` сразу же после команды секционирования, а `\markright` непосредственно перед и после команды секционирования, но первая команда должна быть опущена, если команда секционирования начинает новую страницу.

Типичный случай использования команд `\markboth` и `\markright` связан с применением *-формы команд секционирования. Например, если требуется, чтобы очередная глава не имела номера, её следует начать с команды `\chapter*`. Однако `\chapter*` не только не нумерует главу, но также не заносит её название в верхние колонтитулы, как это делает обычная форма команды `\chapter`.

Чтобы исправить этот недочет, как раз необходимо применить команду `\markboth`:

```
\chapter*{Вместо предисловия}
\markboth{Вместо предисловия}{Вместо предисловия}
```

Следующая декларация присваивает счётчику страниц `page` значение 1 и определяет стиль нумерации страниц.

```
\pagenumbering{num-style}
```

Она имеет глобальную область действия. Аргумент `num-style` должен совпадать с именем одной из команд (за исключением `\fnsymbol`),

которые печатают значения счётчиков, но не иметь обратного слеша:

`arabic` для печати номеров страниц арабскими цифрами,

`roman` — строчными римскими цифрами,

`Roman` — прописными римскими цифрами,

`alph` — строчными латинскими буквами,

`Alph` — прописными латинскими буквами,

`asbuk` — строчными русскими буквами,

`Asbuk` — прописными русскими буквами.

По умолчанию нумерация страниц ведется арабскими цифрами. Печать номеров страниц русскими буквами возможна при загрузке пакета `babel` с опцией `russian`, поскольку команды `\asbuk` и `\Asbuk` определены в этом пакете. Декларация `\pagenumbering{roman}` вводит нумерацию страниц римскими цифрами, а `\pagenumbering{arabic}` восстанавливает арабские цифры. Каждое использование декларации `\pagenumbering` начинает отсчет страниц с единицы, присваивая первый номер текущей странице.

2. Настройка макета

При размещении изображения на листе предусмотрены поля шириной 1 дюйм, т. е. 2,54 см от верхней и нижней кромки листа. Величину полей можно увеличить или уменьшить, изменив значения командных длин

```
\hoffset=2mm     \voffset=-5mm
```

Значения `\hoffset` и `\voffset`, по умолчанию равные нулю, прибавляются к стандартной величине полей, т. е. к 1 дюйму, соответственно, слева и сверху.

Далее приводится набор всех геометрических параметров, управляющих геометрическими параметрами страницы.

`\textheight` — высота текста, т. е. вертикальный размер тела стра-

ницы (без колонтитулов).

`\textwidth` — ширина текста, т. е. горизонтальный размер колонтитулов и тела страницы.

`\oddsidemargin` — расстояние между левым краем листа и левым краем текста на правой странице минус 1 дюйм.

`\evensidemargin` — то же самое для левосторонней страницы. сстояние между верхним краем листа и верхним колонтитулом минус 1 дюйм.

`\headheight` — высота верхнего колонтитула.

`\headsep` — вертикальное расстояние между верхним колонтитулом и телом текстовой страницы.

`\topskip` — минимальное расстояние между верхним краем тела страницы и нижним краем (т. е. базисной линией) первой строки текста. Соответствует

`\baselineskip` (см. ниже) для первой строки страницы.

`\footheight` — высота нижнего колонтитула.

`\footskip` — вертикальное расстояние между нижним краем текста и нижним краем нижнего колонтитула; измеряется между базисными линиями последних строк текста нижнего колонтитула.

`\linewidth` — ширина строки; равна значению `\textwidth` за исключением строк внутри процедур форматирования абзацев, таких как `quote` или `itemize`. Значение `\linewidth` не должно изменяться командами, изменяющими длину.

`\parindent` — ширина отступа в начале абзаца. Внутри парбокса устанавливается равной нулю. Может быть изменена в любом месте.

\baselineskip — минимальное расстояние между нижними краями (базисными линиями) последовательных строк текста. Расстояние между некоторыми строками может быть больше, если они содержат высокие объекты.

Значение \baselineskip устанавливается декларациями, изменяющими размер шрифта. Значение \baselineskip, используемое для всего абзаца, определяется значением, действовавшим на момент завершения абзаца пустой строкой, командой \par или одной из процедур, печатающих текст с абзацного отступа или отдельной строкой. Значение \baselineskip может быть изменено в любом месте входного файла.

Если на странице размещается один абзац, то величина \textheight, деленная на \baselineskip, равна числу строк на странице. \baselinestretch — десятичное число, равное величине интерлиньяжа (межстрочного интервала); по умолчанию равно 1.

Значение \baselinestretch изменяется декларацией \renewcommand. Например, команда

```
\renewcommand{\baselinestretch}{1.25}
```

в преамбуле входного файла приведёт к тому, что весь печатный документ будет напечатан через 1,5 интервала. Чтобы напечатать документ примерно через два интервала, следует задать значение \baselinestretch, равное 1.67. Половине интервала соответствует значение 0.75.

\marginparsep — горизонтальное расстояние между внешним краем текста и заметкой на полях.

\marginparwidth — ширина заметок на полях.

\columnseprule — ширина вертикальной линии между колонками при печати в две колонки. Значение по умолчанию равно нулю, что соответствует невидимой линии.

\columnsep — расстояние между колонками при печати в две колонки.

Параметры настройки, характеризующие горизонтальные размеры, суть нерастяжимые длины. Вертикальные размеры обычно растяжимы; однако расстояние между строками лучше всего выбирать фиксированным. Значение перечисленных параметров настройки можно изменить посредством `\setlength`. Например, если в преамбуле стоит

```
\setlength{\columnseprule}{0.4pt}
```

то при печати в две колонки на каждой странице печатного документа между колонками будет проведена черта толщиной `0,4 pt`.

3. Печать в две колонки

LaTeX форматирует печатный документ в две колонки, если в списке опций `\documentclass` указать `twocolumn`. Опцию `twocolumn` следует использовать, если необходимо напечатать в две колонки большую часть текста. В одноколоночном документе переход к печати в две колонки осуществляет команда

```
\twocolumn[text].
```

Она начинает новую страницу и начинает печать следующего за ней текста в две колонки. Если имеется необязательный аргумент `text`, он будет напечатан в верхней части новой страницы в одну колонку; если опция отсутствует, то печать в две колонки начинается от начала страницы.

Возврат к одноколоночному формату осуществляет команда

```
\onecolumn[text].
```

Она продолжает печатать текст с новой страницы.

Более гибкого форматирования печатного документа позволяет достичь пакет `multicol` из коллекции `tools`.

Описанными выше средствами невозможно сформировать страницу с переменным числом колонок, так как команды `\twocolumn` и `\onecolumn` всегда начинают печатать текст с новой страницы. Кроме того, колонки оказываются несбалансированными, т. е. имеют разную длину.

Пакет `multicol`, написанный Франком Миттельбахом (Mittelbach,

Frank), решает эти проблемы. Он вводит процедуру `multicols`, которая:

- может создавать до 10-ти колонок на произвольном числе страниц;
- выравнивает колонки на последней странице так, чтобы их длины были приблизительно одинаковы;
- может использоваться внутри другой процедуры, такой как `figure` или `minipage`, где она производит бокс с текстом, распределенным на заданное число колонок;
- может проводить между колонками вертикальные разделительные линии заданной толщины;
- имеет удобные средства глобальной и индивидуальной регулировки формата колонок.

Лекция 13

Создание собственных стилевых файлов

Классы и пакеты; команды для создания классов и пакетов.

На этой лекции Вы узнаете, как и зачем создавать собственные стилевые файлы для системы LaTeX 2_ε.

LaTeX представляет собой такую систему подготовки документов, которая позволяет автору сосредоточится собственно на тексте, не слишком заботясь о том, как отформатировать его. Форматирование достигается посредством применения *стилевых файлов*, в которых содержится информация о том, как логическую структуру документа, созданную с помощью команд LaTeX, привести в вид, пригодный для печати.

В актуальной на сегодняшний день версии LaTeX 2_ε понятие стилевых файлов претерпело некоторые изменения. Они были разделены на два подмножества — на *классы (classes)* и *пакеты (packages)*. Это разделение хоть и является во многом условным (физически, и классы и пакеты содержат одно и тоже — команды LaTeX), однако логически оно достаточно однозначно. А именно, пакеты содержат те особенности форматирования, которые не зависят от используемого класса документа.

Кроме этого, существенное отличие от предыдущей версии LaTeX 2.09 заключается в новом наборе специальных команд для написания пакетов и классов, тогда как раньше создатели стилевых файлов вынуждены были довольствоваться низкоуровневыми командами TeX. LaTeX 2_ε предоставляет в распоряжение всем желающим команды высокого уровня. Более того, стало гораздо проще создавать классы и пакеты, унаследованные от уже существующих.

Иногда классы и пакеты, стандартно поставляемые с дистрибутивом, не вполне отвечают запросам авторов. Одно из важных преимуществ LaTeX состоит в том, что автор может настроить под свои потребности практически все аспекты поведения системы. Однако, внесение изменений в эти стандартные классы является плохой практикой, которой следует избегать.

Использование модифицированных стандартных классов лишает документ важного свойства переносимости.

Вместо изменения базовых классов можно создать новый класс на основе одного из них, либо сделать пакет, расширяющий их возможности в соответствии с предъявляемыми требованиями.

1. Создание классов и пакетов

Рассмотрим несколько общих принципов написания собственных классов и пакетов LaTeX.

1.1. Общие принципы

Поддержка стилевых файлов предыдущих версий

По опыту общения с множеством диалектов более ранних версий LaTeX можно утверждать, что поддерживать в течение продолжительного времени стилевые файлы, оставшиеся в наследство от них, практически невозможно. Поэтому старайтесь при первой возможности перейти на новые стандарты LaTeX 2_ε.

Пакет `doc` и утилита `docstrip`

Если Вы собираетесь писать большой класс или пакет для LaTeX, тогда Вам не обойтись без пакета `doc`, который поставляется в комплекте с LaTeX и позволяет в одном файле хранить код и документацию к этому коду одновременно. Из этого файла `doc` может генерировать различные индексы, историю изменений, описания и документацию. В свою очередь, `docstrip` позволяет отделить код от всего остального, и создать собственно файл класса с расширением `.cls`.

Подробное описание `doc` и `docstrip` содержится в файлах `doc.dtx` и `docstrip.dtx` дистрибутива LaTeX 2_ε.

Класс или пакет?

Прежде чем поместить несколько новых команд в стилевой файл необходимо решить, что именно это будет: класс или пакет. Правило, которым

можно руководствоваться, звучит так: *если команды могут использоваться с любым классом, то это пакет, в противном случае это класс!*

Напомним, что существует два типа классов — основные, такие как `article`, `report` или `letter`, и производные, созданные на базе одного из основных классов.

Именование команд

LaTeX включает три типа команд. Во-первых, авторские команды, такие как

```
\section, \texttt  и  \dots,
```

большинство из них достаточно короткие и все буквы в именах в нижнем регистре. Во-вторых, команды для разработчиков классов и пакетов, с длинными именами, состоящими из букв в разных регистрах, такие как

```
\InputIfFileExists,
\RequirePackage,
\PassOptionsToClass.
```

В третьих, внутренние команды TeX, такие как

```
\@tempcnta, \@ifnextchar, \@eha.
```

Большинство таких команд используют символ `@` в имени, т. е. не могут использоваться в документах, только в классах или пакетах. К сожалению, исторически сложилось так, что указанные различия между типами команд достаточно условны. Например, команда `\hbox` — внутренняя, тогда как `\m@ne` это константа -1, которая должна бы обозначаться `\MinusOne`. Тем не менее, настоятельно рекомендуется пользоваться приведенными правилами именования команд как минимум потому, что команды, содержащие символ `@`, не являются частью LaTeX и могут не поддерживаться в будущих версиях.

Боксы и поддержка цвета

Независимо от того, планируете Вы или нет использовать цвета в своих документах, создавая новый класс или пакет Вы должны позаботится

о его совместимости с пакетом `color`, поскольку иначе люди, печатающие документы в цвете, могут быть разочарованы Вашей работой. Самый простой способ обеспечить такую поддержку — всегда использовать команды LaTeX для создания боксов и не использовать для этого примитивы TeX. Другими словами, всегда использовать `\sbox` а не `\setbox`, `\mbox` вместо `\hbox` и `\parbox` или окружение `minipage` взамен `\vbox`.

Определение символов в текстовом и математическом режиме

Поскольку LaTeX 2_ε поддерживает различные кодировки, определения команд, порождающих спецсимволы, акценты и т.д. должны быть сделаны только с использованием предназначенных для этого средств, описанных в документации. Для определения команд такого типа, не зависящих от кодировки, должна использоваться директива `\DeclareRobustCommand`.

1.2. Общие рекомендации

Новая система стилевых файлов предоставляет в Ваше распоряжение достаточно команд для того, чтобы сделать хорошо структурированный класс или пакет, который будет эффективным и переносимым.

Подгрузка других файлов

Следующие команды LaTeX предназначены для использования классов и пакетов внутри других классов или пакетов:

```
\LoadClass        \LoadClassWithOptions
\RequirePackage \RequirePackageWithOptions
```

Настоятельно рекомендуется использовать именно эти команды, а не команду `\input`, по нескольким причинам.

Во-первых, файлы, загруженные при помощи `\input`, не будут отражены в списке `\listfiles`. Во-вторых, если пакет загружается командой `\RequirePackage` (или `\usepackage`), то он будет загружен только один раз, даже если команда встретится несколько раз. В третьих, `\input` не позволяет передать пакету опции при вызове. Таким образом, использо-

вание `\input` может быть оправдано только требованиями совместимости с более ранними версиями.

Эффективность

При создании классов и пакетов считается хорошей практикой использовать там, где это возможно, команды LaTeX. То есть, вместо `\def` рекомендуется использовать `\renewcommand` или `\providecommand`, для определения окружения лучше использовать `\newenvironment` или `\renewenvironment` вместо `\def\foo{...}` и `\def\endfoo{...}`. В случае необходимости изменить значения регистров используйте `\setlength`. Для манипуляций с боксами прибегните к помощи команд `\sbox`, `\mbox` и `\parbox`. Пользуйтесь командами `\PackageError`, `\PackageWarning` или `\PackageInfo` вместо `\@latexerr`, `\@warning` или `\wlog`. Не смотря на то, что по-прежнему можно объявлять опции посредством `\ds@<опция>` и вызова `\@options`, рекомендуется использовать для этого команды `\DeclareOption` и `\ProcessOptions`. Они быстрее работают и используют меньше памяти.

Придерживаясь этих рекомендаций, Вы получите хорошо читаемый код, который, что более важно, с большей вероятностью сохранит работоспособность и в будущих версиях LaTeX.

Переносимость

Другая важная задача при создании стилевого файла — добиться его максимальной переносимости. Чтобы решать эту задачу, необходимо как минимум использовать только отображаемые семибитные символы внутри файла и давать файлам имена, отвечающие правилу 8.3 (восемь символов и имени, символ ., три символа в расширении). Также, естественно, нельзя давать файлам имена, совпадающие со стандартными именами файлов LaTeX, насколько бы они не были близки по содержанию.

Хорошим тоном является использования префикса в именах локальных стилевых файлов. Например, файлы, созданные в Сибирском федераль-

ном университете для собственных нужд могут начинаться с букв `sfu`. Это поможет избежать ситуаций, когда каждый университет будет иметь собственный класс для набора диссертаций с именем `thesis.cls`.

И наконец, если Вы предполагаете использовать определенные возможности ядра LaTeX или какого-то пакета, то будет правильно указать дату, начиная с которой эти возможности поддерживаются. Например, команды вывода ошибок пакета появились в июне 1994 года, так что если Вы намереваетесь их использовать, укажите это явно:

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
```

2. Структура класса и пакета

Классы и пакеты в LaTeX 2_ε более структурированы чем стилевые файлы предыдущих версий. Они состоят из следующих частей:

- *Идентификация.* В этой части указывается что это класс или пакет LaTeX 2_ε и дается краткое его описание.
- *Предварительные определения.* В этой секции определеются некоторые команды и подгружаются другие файлы. Обычно, эти команды необходимы только для следующей секции описания опций.
- *Опции.* В этой секции объявляются и обрабатываются опции.
- *Дополнительные определения.* В этой части решаются основные задачи: объявляются переменные, определяются команды и шрифты, подгружаются другие файлы и т.д.

2.1. Идентификация

Первым делом стилевой файл необходимо идентифицировать. В пакете это делается следующим образом:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{<package>}[<date> <other information>]
```

Например:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{latexsym}%
[1994/06/01 Standard LaTeX package]
```

Идентификация класса происходит похожим образом:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{<class-name>}[<date><other information>]
```

Например так:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{article}%
[1994/06/01 Standard LaTeX class]
```

Все даты должны указываться в формате 'YYYY/MM/DD', отклонение от этого формата может вызывать низкоуровневые ошибки TEX. Эти даты проверяются в случае если пользователь укажет требуемую ему дату создания пакета или класса в командах `\documentclass` или `\usepackage` соответственно.

Описание класса отображается при его подключении, идентификация пакета записывается в лог-файл. Также эти описания отображаются по команде `\listfiles`. Фраза `Standard LaTeX`, приведенная в примере, **запрещена для использования** в пользовательских стилевых файлах, и может быть использована только в файлах дистрибутива LATEX.

2.2. Использование классов и пакетов

Основное отличие между стилевыми файлами версии 2.09 и классами (пакетами) LATEX 2_ε в том, что последние поддерживают модульность, в смысле возможности их разбиения на отдельные файлы, вместо того чтобы использовать один большой файл. Достигается это возможностью использования внутри класса или пакета других пакетов, посредством команды

```
\RequirePackage[<options>]{<package>}[<date>]
```

Эта команда использует тот же синтаксис что и авторская команда `\usepackage`, и позволяет использовать в новом стилевом файле возможности уже существующих пакетов.

В свою очередь, класс может подгружаться другим классом:

```
\LoadClass[<options>]{<class-name>}[<date>]
```

Эта команда имеет тот же синтаксис что и `\documentclass`. Она позволяет классам использовать и расширять возможности друг друга, без необходимости переписывать класс заново.

Кроме того, существуют версии этих же команд, позволяющие подгружать классы и пакеты в точности с теми же самыми опциями, с которыми был подгружен текущий класс.

```
\LoadClassWithOptions{<class-name>}[<date>]
\RequirePackageWithOptions{<package>}[<date>]
```

Например:

```
\LoadClassWithOptions{article}
\RequirePackageWithOptions{graphics}[1995/12/01]
```

2.3. Опции

Другая отличительная особенность новых стилевых файлов LaTeX 2_ε — обработка опций. В классе или пакете можно объявлять опции, которые потом доступны для использования авторами. Заметим, что имена опций должны соответствовать правилам создания имен LaTeX, в частности, не должны содержать никаких контрольных последовательностей символов.

Объявление опции выглядит так:

```
\DeclareOption{<option>}{<code>}
```

Например, в классе `\article`, опция `a4paper` устанавливает размеры страницы по горизонтали и вертикали:

```
\DeclareOption{a4paper}{%
\setlength{\paperheight}{297mm}%
\setlength{\paperwidth}{210mm}}
```

Иногда пользователь указывает опцию, которая не определена в классе или пакете. По умолчанию такая опция вызовет ошибку (в пакете) или предупреждение (в классе). Однако, необъявленные опции также можно обработать следующим образом:

```
\DeclareOption*{<code>}
```

Например, для того чтобы пакет `sfupkg` генерировал предупреждение вместо ошибки, можно использовать такую конструкцию:

```
\DeclareOption*{%
\PackageWarning{sfupkg}%
{Unknown option ‘\CurrentOption’}}
```

Другой пример, это пакет `fontenc`, которые пытается загрузить файл `<ENC>enc.def`, вне зависимости от того, какая кодировка `<ENC>` была передана ему в качестве опции. Это может быть достигнуто такой конструкцией:

```
\DeclareOption*{%
\input{\CurrentOption enc.def}}
```

Существует возможность передать опцию в другой класс или пакет посредством команд `\PassOptionsToClass` или `\PassOptionsToPackage` (имейте ввиду, что эти команды передают только имена опций, но не их объявление!). Например, чтобы передать любую неизвестную опцию в класс `article`, достаточно указать следующее:

```
\DeclareOption*{%
\PassOptionsToClass{\CurrentOption}{article}}
```

Важно отметить, что класс, которому передаются опции (в данном примере `article`), должен загружаться после этой конструкции, иначе опции ему переданы не будут.

Кроме определения опций, о котором говорилось выше, отдельно запускается процесс их обработки, посредством команды

```
\ProcessOptions\relax
```

Эта команда выполняет код, указанный в определении опции, для каждой опции, которая была передана при подгрузке класса или пакета. Например, если пакет `sfupkg` содержит

```
\DeclareOption{foo}{\typeout{Saw foo.}}
\DeclareOption{baz}{\typeout{Saw baz.}}
\DeclareOption*{\typeout{What's \CurrentOption?}}
\ProcessOptions\relax
```

и автор включает в документ команду `[foo,bar]{sfupkg}`, то он увидит следующие сообщения:

```
Saw foo.
What's bar?
```

2.4. Несколько примеров

Минимальный файл класса

В большинстве новых классов и пакетов определяются новые команды, переопределяются старые или каким-либо образом меняются параметры отображения документа. Суть этих действий целиком зависит от воли автора класса или пакета. Однако, существует три вещи, которые обязан содержать любой класс: определение `\normalsize` и значения для `\textwidth` и `\textheight`. То есть минимальный класс должен выглядеть следующим образом:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{minimal}%
[1995/10/30 Standard LaTeX minimal class]
\renewcommand{\normalsize}%
{\fontsize{10pt}{12pt}\selectfont}
\setlength{\textwidth}{6.5in}
\setlength{\textheight}{8in}
```

Отметим, что такой класс сейчас входит в состав дистрибутива LaTeX и называется `\minimal.cls`. Этот класс не поддерживает множества возможностей, без которых трудно представить себе плодотворную работу автора, поэтому большинство других классов содержат гораздо больше этого минимума.

Модифицированный класс `letter`

Любая компания может придерживаться своего стиля в оформлении официальных писем, поэтому ей может понадобиться свой, модифицированный класс `letter` для составления писем. Здесь приводится очень простой пример такого класса, реальный класс был бы существенно больше.

Сначала идентифицируем наш новый класс `sfulet.cls`

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{sfulet}%
[2008/10/01 SFU Press letter class]
```

Далее, передадим все полученные нашим классом опции в базовый класс `letter`, и подгрузим его с опцией `a4paper`.

```
\DeclareOption*{\PassOptionsToClass{%
\CurrentOption}{letter}}
\ProcessOptions\relax
\LoadClass[a4paper]{letter}
```

Чтобы использовать собственное оформление письма, переопределим стиль первой страницы `firstpage`.

```
\renewcommand{\ps@firstpage}{%
\renewcommand{\@oddhead}{<верхний колонтитул>}%
\renewcommand{\@oddfoot}{<нижний колонтитул>}}
```

Мы получили вполне работоспособный новый класс для написания писем, основанный на классе `letter`.

Класс для издания газеты

Простой класс для верстки газеты может быть основан на классе `article`. Объявим этот класс под именем `sfunews.cls`.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{smplnews}%
[1995/04/01 The SFU News newsletter class]
\newcommand{\headlinecolor}{\normalcolor}
```

Он будет передавать все поступающие опции в класс `article`, кроме опции `onecolumn`, которая будет явно отключаться, и опции `green`, которая будет управлять цветом заголовков.

```
\DeclareOption{onecolumn}{\OptionNotUsed}
\DeclareOption{green}%
{\renewcommand{\headlinecolor}{\color{green}}}
\DeclareOption*{\PassOptionsToClass{%
\CurrentOption}{article}}
\ProcessOptions\relax
```

Затем загружается класс `article` с опцией `twocolumn`.

```
\LoadClass[twocolumn]{article}
```

Поскольку газету предполагается печатать в цвете, загружаем пакет `color`.

```
\RequirePackage{color}
```

Далее переопределяем команду `\maketitle`, чтобы сделать заголовок требуемым шрифтом в нужном цвете.

```
\renewcommand{\maketitle}{%
\twocolumn[%
\fontsize{72}{80}\fontfamily{phv}\fontseries{b}%
\fontshape{sl}\selectfont\headlinecolor
\@title
]}
```

Переопределяем `\section`, отключаем нумерацию

```
\renewcommand{\section}{%
\@startsection
{section}{1}{0pt}{-1.5ex plus -1ex minus -.2ex}%
{1ex plus .2ex}{\large\sffamily\slshape\headlinecolor}}
```

И определяем три обязательных параметра

```
\renewcommand{\normalsize}{\fontsize{9}{10}\selectfont}
\setlength{\textwidth}{17.5cm}
\setlength{\textheight}{25cm}
```

На практике создание класса потребует большего, однако в этом примере дан скелет, который может быть взят за основу. Класс `ltnews`, входящий в дистрибутив, не многим сложнее.

3. Команды для создания классов и пакетов

Дадим краткое описание новых команд, которые появились в LaTeX 2_ε специально для написания классов и пакетов.

3.1. Идентификация

Первая группа команд служит для идентификации класса или пакета.

```
\NeedsTeXFormat {<format-name>} [<release-date>]
```

Команда сообщает процессору TeX, что для обработки должен использоваться формат `<format-name>`. Необязательный аргумент `<release-date>` позволяет также указать минимальную требуюмую дату релиза формата. Если релиз будет более ранним, чем указанная дата, то появится соответствующее предупреждение. Стандартное значение `<format-name>` LaTeX 2_ε. Дата, если она указывается, должна быть в формате ‘YYYY/MM/DD’.
Например,

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
```

```
\ProvidesClass{<class-name>}[<release-info>]
```

```
\ProvidesPackage{<package-name>}[<release-info>]
```

Эти команды говорят о том, что файл содержит определения класса `<class-name>` или пакета `<package-name>`. Необязательный аргумент `<release-info>`, если он есть, должен содержать дату релиза в формате 'YYYY/MM/DD' и, возможно, короткое описание и номер версии. Содержание `<release-info>` отображается по команде `\listfiles` и, следовательно, не должно быть очень длинным.
Пример:

```
\ProvidesClass{article}%
[1994/06/01 v1.0 Standard LaTeX class]
\ProvidesPackage{ifthen}%
[1994/06/01 v1.0 Standard LaTeX package]
```

```
\ProvidesFile{<file-name>}[<release-info>]
```

Эта команда похожа на предыдущие, за исключением того, что в качестве обязательного аргумента должно быть указано имя файла с расширением. Используется она для идентификации файлов любых класса или пакета, кроме основного, который идентифицируется одной из предыдущих команд.
Пример:

```
\ProvidesFile{T1enc.def}%
[1994/06/01 v1.0 Standard LaTeX file]
```

Снова заметим, что фраза `Standard LaTeX` не должна использоваться ни в одном файле, не входящем в стандартный дистрибутив LaTeX.

3.2. Подгрузка файлов

Эта группа команд может быть использована для построения своего класса или пакета на базе уже существующих классов и

```
\RequirePackage [<options-list>]%
 {<package-name>} [<release-info>]
```

```
\RequirePackageWithOptions {<package-name>}%
[<release-info>]
```

Команды для загрузки пакетов из других пакетов или классов.
Пример:

```
\RequirePackage{ifthen}[1994/06/01]
\RequirePackageWithOptions{graphics}[1995/12/01]
```

```
\LoadClass[<options-list>]%
          {<class-name>} [<release-info>]
\LoadClassWithOptions{<class-name>} [<release-info>]
```

Эти команды могут использоваться только в классах и только однажды для загрузки другого класса. Они не могут быть использованы в пакетах.
Пример:

```
\LoadClass{article}[1994/06/01]
\LoadClassWithOptions{article}[1995/12/01]
```

Версии `WithOptions` всех перечисленных выше команд просто загружают класс (пакет) с именно теми опциями, с которыми был загружен текущий файл (класс или пакет).

3.3. Опции

Объявление опций

Следующие команды служат для определения и обработки опций классов и пакетов. Любая опция должна удовлетворять правилам именования идентификаторов LaTeX. Существует несколько специальных команд, предназначенных для использования исключительно внутри аргумента `<code>` этих команд (см. ниже).

```
\DeclareOption {<option-name>} {<code>}
```

Делает аргумент `<option-name>` объявленной опцией класса или пакета. После объявления такая опция может быть использована при подключении этого класса или пакета. Аргумент `<code>` содержит код (последовательность команд), который будет выполнен в случае если пакет или класс будет вызван с этой опцией. Код должен состоять из допустимых команд LaTeX.
Пример:

```
\DeclareOption{twoside}{\@twosidetrue}
```

```
\DeclareOption* {<code>}
```

Определяет код, который будет выполнен в случае появления любой опции, с которой был вызван класс или пакет, но которая явно не определена внутри этого класса или пакета.

Если класс не содержит команды `\DeclareOption*`, то все появившиеся неопределенные опции по умолчанию будут передаваться во все пакеты, не генерирую никаких сообщений (впрочем, как и объявленные опции для этого класса).

Если же пакет не содержит конструкции `\DeclareOption*`, то, по умолчанию, любая необъявленная, но появившаяся опция вызовет сообщение об ошибке.

Команды, используемые в коде опций

Следующие две команды могут использоваться только внутри аргумента `<code>` команд `\DeclareOption` или `\DeclareOption*`.

```
\CurrentOption
```

Транслируется в имя текущей опции.

```
\OptionNotUsed
```

Добавляет текущую опцию в список неиспользуемых опций.

В новой версии можно использовать символ решетки (#) внутри кода опции без специальной обработки (в предыдущих версиях требовалось его удваивать).

Передача опций

Приведенные ниже команды также весьма полезны в аргументе <code> команд \DeclareOption и \DeclareOption*:

```
\PassOptionsToPackage{<options-list>}{<package-name>}
\PassOptionsToClass {<options-list>} {<class-name>}
```

Команда \PassOptionsToPackage передает опции из списка <options-list> пакету <package-name>. Это означает, что она добавляет <options-list> к списку опций, с которым позднее будут вызываться команды \RequirePackage или \usepackage для пакета <package-name>.
Пример:

```
\PassOptionsToPackage{foo,bar}{fred}
\RequirePackage[baz]{fred}
```

эквивалентно

```
\RequirePackage[foo,bar,baz]{fred}
```

Подобным образом, \PassOptionsToClass может использоваться в файле класса чтобы передать опции другому классу, который будет загружаться посредством \LoadClass.

Следует четко отличать действие этих двух команд от описанных выше:

```
\LoadClassWithOptions
\RequirePackageWithOptions
```

Команда \RequirePackageWithOptions похожа на \RequirePackage, но она всегда загружает требуемый пакет в точности с тем списком опций, с которым был вызван текущий класс или пакет, а не с теми опциями, которые были переданы явно или при помощи \PassOptionsToPackage.

Главная задача команды \LoadClassWithOptions — позволить легко создавать класс на основе другого, например:

```
\LoadClassWithOptions{article}
```

Сравните эту конструкцию по сложности со следующей:

```
\DeclareOption*{\PassOptionsToClass{%
\CurrentOption}{article}}
\ProcessOptions\relax
\LoadClass{article}
```

Хотя достигаемый результат практически тот же, первая конструкция несомненно лаконичнее, к тому же работает немного быстрее.

Однако, если в классе объявляются собственные опции, то эти конструкции будут давать различные результаты. Сравните, например:

```
\DeclareOption{landscape}{\@landscapetrue}
\ProcessOptions\relax
\LoadClassWithOptions{article}
```

с конструкцией

```
\DeclareOption{landscape}{\@landscapetrue}
\DeclareOption*{\PassOptionsToClass{%
\CurrentOption}{article}}
\ProcessOptions\relax
\LoadClass{article}
```

В первом примере класс `article` будет загружен с опцией `landscape` в том случае, если текущий класс будет вызван с этой опцией. Напротив, во втором примере он никогда не будет загружен с опцией `landscape`, поскольку он вызывается с обработчиком опций по умолчанию, который не передает явно описанные в классе опции.

Код с отложенным исполнением

Первые две команды также предназначены в основном для использования внутри аргумента `<code>` команд `\DeclareOption` и `\DeclareOption*`.

```
\AtEndOfClass {<code>}
\AtEndOfPackage {<code>}
```

Эти команды определяют код, который сохраняется в среде исполнения и вызывается только после обработки всего файла класса или пакета. Повторное использование этих команд не запрещено.

Следующие две команды

```
\AtBeginDocument {<code>}
\ AtEndDocument {<code>}
```

определяют код, которые будет сохранен до тех пор, пока не будут обработаны директивы `\begin{document}` или `\end{document}` соответственно. Код, заданный командой `\AtBeginDocument`, выполняется практически в конце кода `\begin{document}`, после установки таблиц выбора шрифтов. Следовательно, это подходящее место для кода, который должен быть выполнен после того, как все готово для вывода на печать. Однако не рекомендуется использовать сами эти команды для вывода на печать чего либо, так как результаты могут быть непредсказуемы.

Код аргумента `<code>` команды `\AtEndDocument` выполняется в начале кода конструкции `\end{document}`, до того, как будет завершена последняя страница и обработаны не закрытые на этот момент окружения. Если какая-то часть кода должна быть выполнена после этого, то необходимо включить в него инструкцию `\clearpage`. Повторное использование этих команд разрешено. Код сохраняется и выполняется в порядке его появления.

Наконец, команда

```
\AtBeginDvi{<specials>}
```

сохраняет в боксовых регистрах инструкции, которые будут записаны в файл `.dvi` в самое начало первой страницы. Не рекомендуется включать в этот код что-либо, выводящееся на печать. Эту команду также можно использовать неоднократно.

Обработка опций

```
\ProcessOptions
```

Эта команда выполняет <code> для каждой из выбранных опций. Выполняемые ей действия для класса и для пакета различны. Чтобы описать эти отличия определим сначала разницу между глобальными и локальными опциями.

- **Локальные опции** те, которые были явно переданы этому пакету посредством одной из команд

```
\PassOptionsToPackage{<options>}
\usepackage[<options>]
\RequirePackage[<options>]
```

- **Глобальные опции** — все остальные опции, указанные автором в команде `\documentclass[<options>]`.

Например, предположим, что документ начинается так:

```
\documentclass[german,twocolumn]{article}
\usepackage{gerhardt}
```

В свою очередь, пакет `gerhardt` подгружает пакет `fred` следующим образом:

```
\PassOptionsToPackage{german,dvips,a4paper}{fred}
\RequirePackage[errorshow]{fred}
```

тогда:

- локальными опциями пакета `fred` будут `german`, `dvips`, `a4paper` и `errorshow`;
- единственной глобальной опцией пакета `fred` будет `twocolumn`.

При вызове `\ProcessOptions` происходит следующее.

- Во-первых, для каждой опции, объявленной в пакете посредством `\DeclareOption`, проверяется, является ли она локальной или глобальной опцией для этого пакета. Если да, то выполняется код соответствующей опции. Опции проверяются в порядке их объявления в пакете.

- Во-вторых, для каждой из оставшихся локальных опций выполняется команда `\ds@<option>` в случае, если эта опция была где-то объявлена (не в `\DeclareOption`), иначе выполняется код неизвестной опции по умолчанию. Если такого кода не было определено, то генерируется сообщение об ошибке.

В процессе обработки опций система проверяет, чтобы код, объявленный для определенной опции, был выполнен только однажды.

Возвращаясь к примеру, если `fred.sty` содержит следующие определения

```
\DeclareOption{dvips}{\typeout{DVIPS}}
\DeclareOption{german}{\typeout{GERMAN}}
\DeclareOption{french}{\typeout{FRENCH}}
\DeclareOption*{\PackageWarning{fred}%
{Unknown `\CurrentOption'}}
\ProcessOptions\relax
```

то результат обработки этого документа будет следующий:

```
DVIPS
GERMAN
Package fred Warning: Unknown `a4paper'.
Package fred Warning: Unknown `errorshow'.
```

Отметим следующее:

- код опции `dvips` выполняется перед кодом опции `german`, п порядке их объявления;

- код опции `german` выполняется лишь однажды;

- опции a4paper и errorshow вызывают предупреждение, а опция twocolumn — нет, поскольку это глобальная опция.

Поведение \ProcessOptions в классе остается практически таким же, за исключением того, что все опции в классе являются локальными, и значением по умолчанию для \DeclareOption* является \OptionNotUsed, а не сообщение об ошибке.

Заметим, что поскольку \ProcessOptions имеет *-версию, рекомендуется после версии без * всегда ставить \relax, чтобы избежать ненужных ошибок.

```
\ProcessOptions* \@options
```

Эти конструкции работают подобно \ProcessOptions, за исключением того, что опции обрабатываются в порядке из вызова, а не в порядке описания. Вариант \@options сохранен в целях совместимости.

```
\ExecuteOptions {<options-list>}
```

Для каждой из опций, указанной в аргументе, выполняется команда \ds@<option> (если команда не определена, то опция просто игнорируется).

Эта инструкция позволяет определять опции «по умолчанию». Например, \ExecuteOptions{11pt,twoside,twocolumn}.

3.4. Ошибки

Обработка ошибок при работе с файлами

Следующие команды используются в операциях с файлами, они помогают обработать некоторые исключительные ситуации более корректно.

```
\IfFileExists {<file-name>} {<true>} {<false>}
```

Если файл существует, выполняется код аргумента <true>, иначе код аргумента <false>. Отметим, что загрузка файла при этом не осуществляется.

```
\InputIfFileExists {<file-name>} {<true>} {<false>}
```

Если файл существует, то он загружается и непосредственно перед загрузкой выполняется код аргумента `<true>`, если файл не существует, то выполняется аргумент `<false>`.

Сообщения об ошибках

Следующие команды должны использоваться классами и пакетами для вывода сообщений об ошибках или для вывода предупреждений авторам.

```
\ClassError {<class-name>} {<error-text>} {<help-text>}
\PackageError{<package-name>}{<error-text>}{<help-text>}
```

Инструкции для вывода сообщений об ошибках. Выводится сообщение `<error-text>` и стандартное приглашение обработчика ошибок ?. Если пользователь введет в ответ `h`, то будет показан `<help-text>`.

В аргументах `<error-text>` и `<help-text>` могот использоваться команды: `\protect` для предотвращения интерпретации команд, `\MessageBreak` для перевода строки и `\space` для отображения пробелов.

Например, код

```
\newcommand{\foo}{FOO}
\PackageError{ethel}{%
Your hovercraft is full of eels,\MessageBreak
and \protect\foo\space is \foo}{%
Oh dear! Something's gone wrong.\MessageBreak
\space \space Try typing \space <return>
\space to proceed, ignoring \protect\foo.}
```

сгенерирует следующие сообщения

```
! Package ethel Error: Your hovercraft is full of eels,
(ethel) and \foo is FOO.
See the ethel package documentation for explanation.
```

Если пользователь введет h, то появится сообщение

```
Oh dear! Something's gone wrong.
Try typing <return> to proceed, ignoring \foo.
```

```
\ClassWarning {<class-name>} {<warning-text>}
\PackageWarning {<package-name>} {<warning-text>}
\ClassWarningNoLine {<class-name>} {<warning-text>}
\PackageWarningNoLine {<package-name>} {<warning-text>}
\ClassInfo {<class-name>} {<info-text>}
\PackageInfo {<package-name>} {<info-text>}
```

Четыре команды Warning похожи на предыдущие команды, обрабатывающие ошибки, только выводят на экран предупреждение вместо сообщения об ошибках, и не останавливают обработку.

Две команды Info выводят сообщения только в лог-файл. Внутри аргументов <warning-text> <info-text> работают теже инструкции, что и внутри <error-text>.

3.5. Определение команд

LaTeX 2ε предлагает специально для классов и пакетов несколько дополнительных способов (пере)определения команд. Версии этих команд со звездочкой используются для определения команд, которые по меркам TeXне являются длинными.

```
\DeclareRobustCommand{<cmd>}%
[<num>] [<default>] {<definition>}
\DeclareRobustCommand*{<cmd>}%
[<num>] [<default>] {<definition>}
```

Приведенные команды имеют тот же набор аргументов, что и \newcommand, однако они определяют так называемую «грубую» (robust) команду, которая может содержвть внутри определения <definition> опасный код. Ими можно как определить новую, так и переопределить существующую команду, сделав ее «грубой».

Например, если команда \seq определена как

```
\DeclareRobustCommand{\seq}[2][n]{%
\ifmmode
#1_{1}\ldots#1_{#2}%
\else
\PackageWarning{fred}%
{You can't use \protect\seq\space in text}%
\fi  }
```

то она может использоваться даже в перемещаемых аргументах, хотя `\ifmmode` не может. Например,

```
\section{Stuff about sequences $\seq{x}$}
```

```
\CheckCommand{<cmd>}[<num>][<default>]{<definition>}
\CheckCommand*{<cmd>}[<num>][<default>]{<definition>}
```

Эти команды также имеет аргументы как у `\newcommand`, но вместо определдения команды `<cmd>` проверяют, что текущее ее определение в точности совпадает с `<definition>`. Если это не так, то возникает сообщение об ошибке.

3.6. Прочие команды

Параметры разметки страницы

```
\paperheight  \paperwidth
```

Эти параметры обычно устанавливаются классом равными размеру используемой бумаги. Это должен быть реальный размер листа бумаги, в отличие от `\textwidth` и `\textheight`, которые определяют размер текста внутри полей.

Смена регистра

```
\MakeUppercase {<text>}
\MakeLowercase {<text>}
```

TEX предоставляет два примитива `\uppercase` и `\lowercase` для смены регистра текста. К сожалению, эти команды не меняют регистр некоторых символов, например таких как `\ae`. Поэтому в новых классах и пакетах рекомендуется пользоваться приведенными выше командами.

Улучшенные пользовательские окружения в математическом режиме

`\ignorespacesafterend`

Положим, что необходимо определить окружение для отображения текста, пронумерованного как формула в окружении `equation`. Обычный способ это достичь

```
\newenvironment{texteqn}
{\begin{equation}
\begin{minipage}{0.9\linewidth}}
{\end{minipage} \end{equation}}
```

Однако, если это применить в середине параграфа, то появится нежелательный пробел в первой строчке после окружения. Чтобы избежать подобных проблем, и была введена эта новая команда. Используется она так:

```
\newenvironment{texteqn}
{\begin{equation}
\begin{minipage}{0.9\linewidth}}
{\end{minipage} \end{equation}
\ignorespacesafterend}
```

Нормализация пробелов

`\normalsfcodes`

Эта команда используется для восстановления нормальных параметров расстановки пробелов, измененных, например, посредством `\frenchspacing`.

Лекция 14 ⏮

Принципы трансляции исходных файлов. Язык PostScript

Макропакеты для доступа к PostScript из среды LaTeX. Драйверы и утилиты.

1. Принципы трансляции исходных файлов

1.1. Принципы работы TeX

Любой класс, пакет или документ TeX представляет собой текстовый файл, включающий в себя, помимо простого текста, последовательности символов специального назначения, обычно начинающиеся с символа обратной косой черты (`backslash`). Это так называемые «команды» «языка» TeX. В результате обработки входного файла программой TeX, формируется выходной файл формата DVI, который уже не содержит никаких «теховских> команд, а содержит двоичные контрольные последовательности, управляющие процессом вывода текста на экран или печать. Таким образом, TeX является транслятором текстовых файлов определенного формата.

1.2. Трансляция, компиляция и интерпретация

Цель трансляции — преобразовать текст с одного языка на другой, который понятен адресату текста. В случае программ, написанных на языках программирования (`Паскаль, C, BASIC` и т.д.) адресатом является техническое устройство (процессор) или программа-интерпретатор.

Язык процессоров (машинный код) обычно является низкоуровневым. Транслятор, который преобразует программы в машинный язык, принимаемый и исполняемый непосредственно процессором, называется компилятором. Достоинство компилятора: программа компилируется один раз и при каждом выполнении не требуется дополнительных преобразований. Соответственно, не требуется наличие компилятора на целевой машине, для которой компилируется программа. Недостаток: отдельный этап компиля-

ции замедляет написание и отладку и затрудняет исполнение небольших, несложных или разовых программ.

Противоположный метод реализации — когда программа исполняется с помощью интерпретатора, вообще без трансляции. Интерпретатор программно моделирует машину, цикл выборки-исполнения которой работает с командами на языках высокого уровня, а не с машинными командами. Такое программное моделирование создаёт виртуальную машину, реализующую язык. Этот подход называется чистой интерпретацией. Чистая интерпретация применяется как правило для языков с простой структурой. Интерпретаторы командной строки обрабатывают команды в скриптах в UNIX или в пакетных файлах (`.bat`) в `MS-DOS` также, как правило, в режиме чистой интерпретации.

Достоинство чистого интерпретатора: отсутствие промежуточных действий для трансляции упрощает реализацию интерпретатора и делает его удобнее в использовании, в том числе в диалоговом режиме. Недостаток — интерпретатор должен быть в наличии на целевой машине, где должна исполняться программа. А свойство чистого интерпретатора, что ошибки в интерпретируемой программе обнаруживаются только при попытке выполнения команды (или строки) с ошибкой, можно признать как недостатком, так и достоинством.

Существуют компромиссные между компиляцией и чистой интерпретацией варианты реализации языков программирования, когда интерпретатор перед исполнением программы транслирует её на промежуточный язык, более удобный для интерпретации (то есть речь идёт об интерпретаторе со встроенным транслятором). Такой метод называется смешанной реализацией. Примером смешанной реализации языка может служить Perl. Этот подход сочетает как достоинства компилятора и интерпретатора (большая скорость исполнения и удобство использования), так и недостатки (для трансляции и хранения программы на промежуточном языке требуются дополнительные ресурсы; для исполнения программы на целевой машине должен быть представлен интерпретатор). Также, как и в случае компилятора, смешанная реализация требует, чтобы перед исполнением исходный

код не содержал ошибок (лексических, синтаксических и семантических).

Трансляция и интерпретация — разные процессы: трансляция занимается переводом программ с одного языка на другой, а интерпретация отвечает за исполнение программ. Однако, поскольку целью трансляции как правило является подготовка программы к интерпретации, то эти процессы обычно рассматриваются вместе. Например, языки программирования часто характеризуются как «компилируемые» или «интерпретируемые», в зависимости от того, преобладает при использовании языка компиляция или интерпретация.

С другой стороны, существует взаимопроникновение процессов трансляции и интерпретации: интерпретаторы могут быть компилирующими, а в трансляторах может требоваться интерпретация для конструкций метапрограммирования (например, для макросов в языке ассемблера, условной компиляции в `Си` или для шаблонов в `C++`).

Более того, один и тот же язык программирования может и транслироваться, и интерпретироваться, и в обоих случаях должны присутствовать общие этапы анализа и распознавания конструкций и директив исходного языка.

1.3. Формат DVI

Итак, TEXпредставляет собой транслятор с исходного языка (который тоже называется TEX, что иногда вызывает путаницу) на язык, управляющий выводом текста и графики на экран или принтер. Этот язык называется `DVI` (от англ. `DeVice Independent`).

Здесь необходимо отметить, что язык программирования TEX является полным по Тьюрингу, т.е. на нем можно реализовать любую вычислимую функцию. Об этом речь пойдет далее.

Файлы `DVI` не приспособлены для чтения человеком; они содержат двоичные данные, описывающие визуальное представление документа способом, не ориентированным на какой-либо формат изображения, монитор или принтер (отсюда название формата). Файлы `DVI` обычно подаются на вход другой программы (называемой `DVI`-драйвером), которая преобразует

их в графические данные. Например, многие дистрибутивы TEX включают программу предпросмотра файлов DVI на мониторе — эта программа является драйвером. Драйверы также используются для конвертирования файлов DVI в другие форматы (такие как PostScript, PDF) и для печати. О формате (или, правильнее, о языке) PostScript мы поговорим более подробно.

2. Язык PostScript

PostScript (Постскрипт) — язык описания страниц, в основном используемый в настольных издательских системах.

2.1. Общее описание языка

Postscript был разработан Джоном Уорноком и Чаком Гешке из Adobe Systems в начале 80-х гг. Исходно Postscript использовался как ядро механизма печати компьютеров Apple, но вскоре стал широко распространенным стандартом для большинства компьютерных систем. Интерпретаторы Postscript (в виде программных или аппаратных компонентов) для печати документов присутствуют практически во всех современных компьютерных системах. В Postscript используется модель изображения текста (или рисунков) на чистой странице. Когда страница готова, она выводится на печать и начинается «прорисовка» изображения очередной страницы.

В 1984 году увидел свет первый вариант PostScript (позже к названию было добавлено Level 1, чтобы отличать его от Level 2). Postscript имел огромные преимущества перед другими системами того времени:

- Платформонезависимость. Один и тот же файл мог печататься как на лазерном принтере, выдававшем тогда 300 dpi, так и на фотонаборном устройстве с 2400 dpi с наилучшим качеством в каждом случае.

- Любой производитель мог лицензировать интерпретатор PostScript и использовать PostScript со своим устройством.

- Спецификации PostScript были общедоступны, таким образом, любой разработчик мог писать программы, поддерживающие PostScript.

В 1991 Adobe выпустила следующую версию PostScript — PostScript Level 2. Основными улучшениями были:

- Увеличенная скорость и надёжность (в основном это коснулось управления памятью).
- Поддержка цветоделения в самом контроллере.
- Распаковка сжатых изображений в самом контроллере (JPEG и CCITT group 4).
- Поддержка больших наборов шрифтов (для азиатских стран).
- Кэширование шрифтов и изображений.
- Улучшенные драйверы и алгоритмы печати изображения.

В 1998 году Adobe ввела новый стандарт PostScript 3. Изменения по сравнению с Level 2 незначительны.

- Поддержка 4096 уровней на один цвет.
- Поддержка PDF. Контроллеры PS Level 3 могут выводить PDF наряду с файлами PS Level 2.
- Расширенная поддержка цветоделения. Контроллеры PS 3 могут производить цветоделение изображений, заданных в оттенках серого или в шести цветах.
- Добавлена интернет-функциональность (Web-ready printing). К сожалению, пока не реализована.

2.2. Применение PostScript

PostScript соединил в себе лучшие возможности растровой и векторной графики. С одной стороны, PostScript предоставляет возможность вывода высококачественной векторной графики и единый язык управления, который может быть использован любым производителем принтеров. С другой, PostScript предлагает удобные возможности по печати растровой графики и текста. А главное его достоинство в том, что PostScript может совмещать все эти типы вывода на одной странице.

PostScript — больше, чем типичный язык управления принтером, он является полнофункциональным языком программирования. Многие прикладные программы могут преобразовать документ в PostScript-программу, при выполнении которой будет получен начальный документ. Эта программа может быть послана непосредственно на принтер с поддержкой PostScript или преобразована интерпретатором PostScript в другой формат (для принтеров без поддержки PostScript), или результат её выполнения интепретатором может быть показан на экране. Так как исходная PostScript-программа одна и та же, PostScript называется независимым от устройства.

Большинство высокопроизводительных принтеров и плоттеров имеют встроенный интерпретатор языка PostScript.

2.3. Краткая характеристика языка

PostScript — полнофункциональный язык программирования. Хотя программы на PostScript и создаются в основном не людьми, а другими программами, в принципе ничто не мешает разрабатывать на нем практически любые программы.

PostScript — интерпретируемый стековый язык. Синтаксис языка использует обратную польскую нотацию, что делает ненужным использование скобок, однако требует некоторой практики для чтения текста программы из-за необходимости держать в голове содержимое стека. Большинство операторов берут операнды со стека и помещают результат вычислений в стек. Литералы (строки и числа) помещают свою копию в стек.

Структура файла PostScript формируется структурными комментариями. Структурные комментарии начинаются с последовательности символов %% в начале строки.

Файл PostScript состоит из четырех частей: заголовка, пролога, тела и эпилога.

Заголовок начинается строкой %!PS-Adobe-N.M, где N.M — версия спецификации, далее идут структурные комментарии с общими свойствами документа, и заканчивается заголовок строкой %%EndComments.

В прологе обычно содержатся описания подпрограмм и данные, необходимые для печати документа — например, процедуры и шрифты. Пролог заканчивается комментарием %%EndProlog.

В теле программы содержатся команды, отвечающие за отрисовку каждой страницы, предваряемые строкой %%Page: <метка> <номер>, где <метка> — метка страницы, которая не выводится на печать, и <номер> — порядковый номер страницы в файле.

После команд отрисовки всех страниц следует структурный комментарий %%Trailer, после чего начинается эпилог.

В эпилоге можно продублировать комментарии из заголовка (это реализовано на случай, если в начале вывода программе ещё не известно, например, число страниц) — в этом случае вместо числовых значений параметров в заголовке надо писать atend).

Заканчивается файл строкой %%EOF.

2.4. Обзор элементов языка

Стеки

В PostScript для хранения данных разных типов используются четыре стека: стек операндов, стек графических контекстов, стек исполнения, стек словарей.

Типы данных

Основные типы данных языка PostScript: целое число со знаком, вещественное число, логическое выражение, строка символов, массив, процедура, словарь, имя.

Операторы

Операторы в PostScript получают операнды (параметры) с вершины стека, и записывают результаты действий (если они есть) также в стек. Такой стиль программирования, при котором операнды задаются до операции над ними (оператора), называется постфиксной нотацией. Например, для сложения двух заданных чисел необходимо вначале поместить их в вершину стека операндов, затем выполнить команду add, которая извлечет из стека два элемента, помещенные последними (то есть заданные числа), сложит их и запишет результат в стек, откуда он сможет быть извлечен другой операцией в качестве операнда. Это записывается так:

```
4 5 add
```

Операторы PostScript можно разделить на несколько групп. Это операторы для работы с содержимым стека (независимо от типа), арифметические операции, операторы работы с массивами, работы со словарями, работы со строками, сравнения, управления потоком команд, работы с типом аргументов и преобразования, доступа и работы с файлами, работы с виртуальной памятью, графического состояния и другие. Описания операторов PostScript можно найти в документации по языку.

2.5. Пример программы

```
%!PS-Adobe-2.0
/Times-Italic findfont % установить шрифт Times курсив
30 scalefont           % размером 30 пунктов
setfont

/printZip {                    % процедура вывода текста
0 0 moveto                     % перейдем в точку 0 0
(Can you see a shadow ?) show% выведем на экран текст
} def                          % конец описания процедуры
30 500 translate  % установить начало координат
```

```
.95 -.05 0 {      % цикл от 0,95 до 0 с шагом -0,05
setgray           % установить цвет в оттенок серого,
            %равный текущему значению параметра цикла
printZip          % вызвать процедуру вывода на экран
-.5 .5 translate % сместить начало координат
} for             % конец цикла

1 setgray printZip% вывести текст белым цветом

showpage           % вывести страницу
```

Если сохранить приведенный выше текст в файле с расширением `.ps` и просмотреть или вывести на печать, то должна получиться примерно такая картинка:

Рис. 14.1. Результат работы программы

3. Поддержка языка PostScript в TEX. Пакет PSTricks

Пакет `PSTricks` представляет собой набор команд TEX, которые позволяют после обработки транслятором TEX получать на выходе инструкции языка `PostScript`. Эти инструкции включаются в `DVI`-файл, и, после обработки утилитой `dvips`, попадают в результирующий `ps`-файл. Таким образом, `PSTricks` позволяет с одной стороны включать в свои TEXовские документы довольно сложные графические элементы, а с другой — создавать средствами TEX графические файлы на языке `PostScript`.

Далее приводится минимум основных сведений, необходимых для работы с пакетом `PSTricks`. За более подробной информацией можно обратиться к документации по `PSTricks`, которая вместе с самим пакетом включена в стандартный дистрибутив LATEX 2_ε.

3.1. Аргументы и разделители

PSTricks использует следующие виды разделителей:

- Фигурные скобки `{аргумент}`
- Квадратные скобки (для необязательных аргументов) `[аргумент]`
- Круглые скобки и запятые для координат `(x,y)`
- Знак равенства и запятые для параметров `параметр1=значение1, ...`

В качестве разделителя между челой и дробной частью десятичных чисел всегда используется точка, а не запятая, поскольку запятая имеет специальное значение. Самые распространенные ошибки при использовании PSTricks — неверное использование разделителей. Типичные ошибки, которые возникают при этом, такие

```
! Use of \get@coor doesn't match its definition.
! Paragraph ended before \pst@addcoor was complete.
! Forbidden control sequence found while scanning use of
  \check@arrow.
! File ended while scanning use of \lput.
```

Поскольку макросы `PSTricks` используют множество аргументов, полезно помнить, что между аргументами можно вставлять пробелы или символы конца строки (для лучшей читаемости), кроме аргументов, заключенных в фигурные скобки. Если нужно вставить пустую строку между аргументами в фигурных скобках, то обязательно следует поставить в конце строки знак комментария `%`.

Как правило, в общем случае, первым непустым символом после команды PSTricks не должны быть `[` или `(`. Иначе PSTricks может распознать эти символы как часть команды. Конечно, всегда можно обойти это ограничение, вставив пару скобок `{}` между командой и `[` или `(`.

3.2. Цвет

По умолчанию в PSTricks присутствуют цвета

```
red, green, blue, cyan, magenta, yellow
```

и оттенки серого

```
black, darkgray, gray, lightgray, white,
```

Это означает, что эти имена могут использоваться в командах, описанных ниже. Также это означает, что определены команды \red, grey и т.д., которые можно использовать для смены цвета практически также, как и \rm или \tt для смены начертания шрифта. Ниже приведены несколько отличий команд цвета от шрифтовых команд.

- Команды цвета могут использоваться как в математическом режиме так и вне его, обязательно лишь использовать правильную группировку TEX.
- Команды смены цвета влияют на все, что попадает в область их действия, а не только на символы.
- Область действия команд управления цветом не может распространяться между страницами (перейти на другую страницу).
- Команды смены цвета не так безболезненно используются в определениях боксовых команд. Однако, большинства проблем можно избежать просто явно группирую команды управления цветом (т.е. заключая их область действия в фигурные скобки).

Можно определять собственные цвета или переопределять существующие при помощи следующих команд.

```
\newgray{<color>}{<num>}
```

Аргумент <num> указывает яркость серого цвета в терминах PostScript, где для черного цвета используется значение 0, а для белого 1. Например,

```
\newgray{darkgray}{.25}
```

```
\newrgbcolor{<color>}{<num1 num2 num3>}
```

Аргумент `<num1 num2 num3>` задает значение RGB-цвета в спецификации PostScript, где каждая компонента может принимать значение от 0 до 1. Например,

```
\newrgbcolor{green}{0 1 0}
```

```
\newhsbcolor{<color>}{<num1 num2 num3>}
```

Аргумент `<num1 num2 num3>` задает значение HSB-цвета в спецификации PostScript, где каждая компонента может принимать значение от 0 до 1. Например,

```
\newhsbcolor{mycolor}{.3 .7 .9}
```

```
\newcmykcolor{<color>}{<num1 num2 num3 num4>}
```

Аргумент `<num1 num2 num3 num4>` задает значение CMYK-цвета в спецификации PostScript, где каждая компонента может принимать значение от 0 до 1. Например,

```
\newcmykcolor{hercolor}{.5 1 0 .5}
```

Рекомендуемой моделью для задания нового цвета является RGB. Модель CMYK, хотя и является лучшей для цветной печати, однако поддерживается не всеми реализациями PostScript Level 1. Использовать модель HSB вообще не рекомендуется.

3.3. Установка графических параметров

PSTricks использует запись `параметр=значение` для изменения графических параметров. Изменять значения параметров можно командой `\psset`

```
\psset{fillcolor=yellow}
\psset{linecolor=blue,framearc=.3,dash=3pt 6pt}
```

Синтаксис команды следующий:

```
\psset{<par1=value1,par2=value2,...>}
```

Как показано в предыдущем примере, пробел в некоторых случаях служит разделителем внутри значения параметра. Кроме этого, пробел может стоять после запятой, разделяющей параметры.

Практически все команды позволяют указывать пары **параметр=значение** в качестве первого необязательного аргумента в квадратных скобках.

```
\psline[linecolor=green,linestyle=dotted](8,7)
```

полностью эквивалентно

```
{\psset{linecolor=green,linestyle=dotted}\psline(8,7)}
```

Для многих параметров PSTricks вычисляет значение и хранит его в специальном формате, готовом для передачи в PostScript. Другие хранятся в виде значений переменных TEX, и могут использоваться для подстановки в качестве значений других параметров. К примеру

```
\psset{linecolor=\psfillcolor,doublesep=.5\pslinewidth}
```

Однако, даже эти параметры PSTricks может обрабатывать специальным образом, проверяя, например, на наличие ошибок, поэтому настоятельно не рекомендуется переопределять команды TEX, хранящие значения таких параметров.

3.4. Размеры, координаты и углы

Если аргумент команды PSTricks является размером, то указывать единицы измерения не обязательно, поскольку в пакете заданы единицы измерения по умолчанию. Конечно, их можно изменить, установив параметр

```
unit=<dim>
```

Если никогда не указывать единицы измерния, то можно изменять масштаб рисунка просто задавая параметр `unit`. Использовать единицы измерения по умолчанию можно и для установки размеров, не относящихся непосредственно к PSTricks. Сделать это можно посредством команд

```
\pssetlength{<cmd>}{<dim>}
\psaddtolength{<cmd>}{<dim>}
```

Эти команды работают аналогично командам LaTeX \setlength и \addtolength.

Координаты записываются в виде пары (x,y). Начало координат совпадает с текущей координатой TeX. Команда \SpecialCoor позволяет использовать полярные координаты в форме (r;a), где r — радиус, а a — угол.

Параметр unit фактически устанавливает три параметра

```
xunit=<dim> (Default: 1cm)
yunit=<dim> (Default: 1cm)
runit=<dim> (Default: 1cm)
```

Это единицы измерения для координат по оси x, оси y и остальных координат соответственно. Они хранятся в регистрах \psunit (и \psrunit), \psxunit, \psyunit.

```
\degrees[<num>]
```

Аргумент <num> указывает количество единиц на круге. Так, команда \degrees без параметров аналогична команде \degrees[360], а команда \radians ни что иное, как другая форма записи для \degrees[6.28319].

3.5. Основные графические параметры

Толщина и цвет линий задаются параметрами

```
linewidth=<dim>     Default: .8pt
linecolor=<color>   Default: black
```

Параметр linewidth хранится в регистре \pslinewidth, параметр linecolor в регистре \pslinecolor.

Области, ограниченные кривыми, допускают заливку в соответствии с параметрами

```
fillstyle=<style>
fillcolor=<color>
```

Значение `fillstyle=none` означает отсутствие заливки, `fillstyle=solid` сплошная заливка цветом `fillcolor`. Другие стили заливки подробно описаны в документации к пакету.

Все команды рисования графических объектов имеют версию «со звездочкой», например `\psframe*`, которые воспроизводят данный объект сплош залитый цветом `linecolor`.

Незамкнутые кривые могут снабжаться стрелками на концах, в соответствии со значением параметра

```
arrows=<arrows>
```

Значение `arrows=-` означает отсутствие стрелок, значения `arrows=<->`, `arrows=->`, `arrows=<-` означают стрелки на обеих или на одном из концов соответственно. Для незамкнутых кривых желаемое расположение стрелок можно указать и в качестве аргумента в фигурных скобках, например

```
\psline[linewidth=2pt]{<-}(2,1)
```

Другие виды стрелок описаны в документации.

Лекция 15 ⏮

Создание новых команд. Программирование на языке TEX

Команды без аргументов, команды с аргументами, создание новых процедур.

Как уже упоминалось выше, TEX (как язык программирования) является функционально полным, т.е. на нем можно реализовать любую вычислимую функцию. Среди всех возможностей языка для создания пригодных для печати документов используется лишь малая часть, связанная непосредственно с набором. Это определение и переопределение команд, счетчиков, размеров и пр. Это вполне оправдано, поскольку позволяет автору, пользующемуся TEX, не быть программистом. Да и у программистов обычно не возникает желания использовать TEX для разработки, например, информационных систем. Однако, для создания собственного формата или сложного стиля и для решения других технических задач знание этих возможностей не будет лишним.

1. Определения и переопределения в LATEX

Формат LATEX отличается от многих других форматов TEX тем, что в нем реализована идея, высказывавшаяся Д. Кнутом в его книге «The TEXbook», о создании системы, основанной на TEX, однако замкнутой в отношении множества команд, т.е. такой, что любая ее функция реализуется только посредством ее собственных команд, не используя примитивы TEX. Соответственно, для определения и переопределения объектов LATEX также использует собственные команды, о которых и пойдет дальше речь.

1.1. Команды

Для определения команд используется конструкция

```
\newcommand{<cmd>}[<args>]{<definition>}
\newcommand{<cmd>}[<args>][<default>]{<definition>}
```

<cmd> определяет имя новой команды, и должно начинаться с символа \. Количество аргументов команды поределяется целым числом <args>. Максимальное значение равно 9. По умолчанию команда определяется без аргументов. Значение аргументов по умолчанию задается необязательным параметром <default>. Определение <definition> содержит текст, которым будет заменяться встреченная в документе команда \cmd. В определении команды параметры в количестве, заданном <args>, обозначаются символом # с идущим за ним номером аргумента, от 1 до 9 соответственно. Если команда \cmd уже определена, будет выдано сообщение об ошибке. Для переопределения уже существующих команд используются

```
\renewcommand{<cmd>}[<args>]{<definition>}
\renewcommand{<cmd>}[<args>][<default>]{<definition>}
```

соответственно, которые имеют точно такой же синстаксис.

1.2. Счетчики

Команда

```
\newcounter{<name>}[<counter>]
```

создает счетчик с именем name и со значением по умолчанию равным 0. Необязательный параметр <counter> позволяет сослаться на другой существующий счетчик с именем counter. В этом случае значение нового счетчика будет сбрасываться в 0 при увеличении значения counter.

1.3. Окружения

Окружения позволяют ограничивать часть текста, который должен быть отформатирован способом, существенно отличающимся от

```
\newenvironment{<name>}[<args>]{<begdef>}{<enddef>}
\newenvironment{<name>}[<args>][<default>]%
                {<begdef>}{<enddef>}
\renewenvironment{<name>}[<args>]{<begdef>}{<enddef>}
```

<name> указывает на название нового либо переопределяемого окружения, целое число <args> определяет количество аргументов у окружения (по умолчанию нет аргументов). Текст <begdef> заменяет появившуюся в

документе инструкцию `\begin{name}`, внутри можно использовать имена параметров, которые обозначаются парами `#<номер параметра>`, например `#1`. `<enddef>` подставляется соответственно вместо `\end{name}`, использовать здесь параметры нельзя.

1.4. Шрифты

Для определения новых имен шрифтов используется команда

```
\newfont{<cmd>}{<font_name>}
```

`<cmd>` — имя новой команды, которая ранее не должна быть определена. `<font_name>` — наименование шрифта, который будет выбран при появлении в документе команды `\cmd`.

1.5. Размеры

```
\newlength{\<name>}
```

определяет новый размер с именем `<name>` и значением 0 по умолчанию. Если команда с таким именем уже определена, то возникнет сообщение об ошибке.

2. Макросы TEX

Макрос (или макроподстановка) позволяет заменять одну или несколько команд одной командой. Приведенные выше конструкции решают подобные задачи, однако они сами являются макросами, написанными с использованием примитивов TEX. Далее мы будем рассматривать возможности, предоставляемые базовой системой TEX.

2.1. Упрошение ввода длинных последовательностей команд

Предположим, что в тексте часто встречается какая-то последовательность инструкций, например это может быть обозначение точки n-мерного пространства $(x_1, \ldots, x_n)$, которое в файле выглядит как `(x_1, \dots, x_n)`. TEX

```
\def\<name>[<params>]{<definition>}
```

Название новой команды задается аргументом `<name>`. Необязательный аргумент `<params>` задает количество и формат входных параметров новой команды, которые обозначаются символом `#` и цифрой от 1 до 9. Определение новой команды `<definition>` содержит текст, который TEX будет подставлять всегда, когда встретит команду `\<name>`. Внутри `<definition>` можно использовать параметры, перечисленные в `<params>`. Причем если в `<params>` параметры должны перечисляться обязательно по порядку, то в `<definition>` они могут использоваться несколько раз и в произвольном порядке.

Вернемся к примеру с n-мерным вектором. Для облегчения набора мы можем определить команду `\vec` как

```
\def\vec{(x_1, \dots, x_n)}
```

Это будет работать, но допустим мы пожелаем использовать в тексте обозначение не одной, а двух или более точек. Тогда нам потребуется использовать параметр вместо x

```
\def\vec#1{(#1_1, \dots, #1_n)}
```

Если же предположить, что и пространств будет больше одного, то имеет смысл ввести второй параметр и окончательно записать определение как

```
\def\vec#1#2{(#1_1, \dots, #1_#2)}
```

Итак, вместо макроса TEX будет подставлять его определение, или, как иногда говорят, будет раскрывать макрос. Однако, если в состав макроса входят другие макросы, то TEX начнет раскрывать их и так до тех пор, пока не останутся только примитивы TEX.

2.2. Синонимы

Кроме макроопределений TEX имеет еще один способ определения новых команд. Это примитив

```
\let\<newname>=<definition>
```

где <newname> это любое допустимое имя, а <definition> может быть любым символом или командой. В результате такого определения новая команда будет эквивалентна указанному определению, т.е фактически будет являться синонимом. Например, если определить \let\a=\def, то можно будет использовать \a для определения макросов.

Может показаться что \let работает также как и \def, но то не совсем так. Различие состоит в том, что \def определяет команду, которую TEX при обработке заменит, а \let определяет синоним, фактически копирует команду или символ. Поэтому и поведение новых команд будет разное. Например, определим \def\a{1,2,3,\dots} и \let\b=\a. Команды \a и \b будут работать одинаково, причем \b не будет раскрываться сначала в \a а потом в определение \a, а сразу раскроется в определение \a. А если мы определим \def\one{1} и \let\one=1, то конструкция, например, \vskip\one pt в первом случае сработает, а во втором вызовет сообщение об ошибке, поскольку \vskip будет искать цифру а не команду.

2.3. Проверка условий. Ветвление

Для реализации любого алгоритма на языке программирования необходимо чтобы в этом языке поддерживались три базовые алгоритмические конструкции — линейная, разветвляющаяся и циклическая. TEX поддерживает их все, и этот раздел посвящен рассмотрению конструкций, обеспечивающих ветвление. Этих конструкций несколько, общая их форма такова

```
\if<condition> <true text>\else <false text>\fi
```

Если логическое выражение <condition> истинно, то будет пропущен текст <false text>, если оно ложно, то текст <true text>. Если текст <false text> отсутствует, то и команда \else также может быть опущена. Первая часть конструкции ветвления представляет собой команду, начинающуюся с \if, после чего, без пробелов или других разделителей, следует условие <condition>. Фактически это одна команда, поэтому можно сказать что TEX поддерживает множество комад для организации

ветвления. Более того, пользователь также может определять такие команды, что и будет показано ниже. Приведем пример:

```
\ifodd\count0 Четное\else Нечетное\fi
```

Эта конструкция проверяет значение регистра `\count0` в зависимости от его четности выводит соответствующее сообщение. Приведем перечень команд проверки условий, которые поддерживает TEX.

```
\ifnum<num1><relation><num2>   (сравнение двух чисел)
\ifdim<dim1><relation><dim2>(сравнение двух размеров)
\ifodd<number>           (проверка четности <number>)
\ifvmode             (проверка на вертикальный режим)
\ifhmode           (проверка на горизонтальный режим)
\ifmmode           (проверка на математический режим)
\ifimode               (проверка на внутренний режим)
\if<token1><token2>    (проверка на совпадение кодов)
\ifcat<token1><token2> (проверка на совпадение
                                          категорий)
\ifx<token1><token2> (проверка на соответствие
                                           объектов)
\ifvoid<num>                (проверка регистра бокса)
\ifvbox<num>                (проверка регистра бокса)
\ifhbox<num>                (проверка регистра бокса)
\ifeof<num>                 (проверка на конец файла)
\iftrue                     (всегда истина)
\iffalse                    (всегда ложь)
```

TEX поддреживает возможность определения собственных конструкций `\if...`, при помощи команды `\newif`. При этом фактически определяются три команды, одна для проверки условия, вторая для обращения его в истину и третья, для обращения его в ложь. Например, команда `\newif\ifmytest` определяет команды `\ifmytest`, `\mytesttrue` и `\mytestfalse`.

2.4. Циклы

Для полноты арсенала TEX как языка программирования не хватает только циклов, выполняющих повторяющиеся действия. О них сейчас и пойдет речь. TEX содержит конструкцию `\loop ... \repeat`, которая выглядит так:

```
\loop \if<условие> <действия2>\repeat
```

Сначала выполняются инструкции `<действия1>`, потом проверяется условие `<условие>`. Если оно истинно, то выполняются инструкции `<действия2>` и все начинается с начала, в противном случае цикл завершается. Обратим внимание на то, что условная конструкция `\if...` не имеет пары `\fi`, она закрывается командой `\repeat`.

2.5. Операции с файлами

TEX может одновременно читать строки из 16 файлов, посредством следующих операций:

```
\openin<number>=<filename>
```

открывает файл `<filename>` для чтения и ассоциирует с ним номер `<number>`, который будет использоваться в остальных файловых операциях.

```
\closein<number>
```

закрывает файл, связанный с номером `<number>`.

```
\read<number> to <cmd>
```

читает из файла, ассоциированного с номером `<number>`, очередную строку и помещает ее в команду `<cmd>`, которая по сути является макросом без параметров, раскрывающимся в эту строку при появлении в тексте. Если `<number>` меньше нуля или больше 15, то для чтения используется устройство стандартного ввода, для отрицательных значений не выводится текст приглашения для ввода.

2.6. Арифметические операции

Для изменения значений целых переменных, объявляемых командой `\newcounter`, поддерживаются четыре основыние арифметические операции. Команды TEX для их выполнения следующие:

```
\advance <num1> by<num2> %сложение и вычитание
\multiply <num1> by<num2> %умножение
\divide <num1> by<num2> %деление
```

Эти примитивы выполняют арифметическую операцию, использую `<num1>` в качестве левого операнда, `<num2>` в качестве правого и помещают результат в `<num1>`.

2.7. Пример

Приведем пример из книги Д. Кнута «The TEXbook», описывающий макрос `\primes{n}`, который раскрывается в список первых n простых чисел.

```
\newif\ifprime\newif\ifunknown % булевские переменные
\newcount\n\newcount\p\newcount\d \newcount\a  %целые
\def\primes#1{2,~3
             % предполагаем, что #1 как минимум равен 3
\n=#1 \advance\n by-2              % количество чисел n
\p=5                           % первое простое больше 3
\loop\ifnum\n>0 \printifprime\advance\p by2 \repeat}
\def\printp{,             % макрос печати простого числа
  \ifnum\n=1 и~\fi %перед последним числом печатаем 'и'
  \number\p \advance\n by -1 }
\def\printifprime{\testprimality \ifprime\printp\fi}
\def\testprimality{{\d=3 \global\primetrue
\loop\trialdivision \ifunknown\advance\d by2 \repeat}}
\def\trialdivision{\a=\p \divide\a by\d
 \ifnum\a>\d \unknowntrue\else\unknownfalse\fi
  \multiply\a by\d
  \ifnum\a=\p \global\primefalse\unknownfalse\fi}
```

Раздел 3. Программирование презентационных эффектов

Лекция 16 ⏮

Управление цветом, графический драйвер

Выбор драйвера, пакет color, выбор цвета. Цветные боксы, фон страницы и таблицы.

Поддержка цвета в LaTeX обеспечивается пакетом `color`. Цветные таблицы позволяет создавать пакет `colortbl`. Далее рассматриваются эти два пакета.

1. Пакет color

1.1. Драйвер цветной печати

Упомянутые пакеты основаны на возможностях, которые не относятся непосредственно к TeX. Эти возможности поддерживаются так называемым драйвером, продназначенным для печати файла `dvi`. К сожалению, не все драйверы поддерживают одинаковые возможности, так что графические пакеты обычно имеют опции, в которых указывается какой драйвер нужно использовать. Перед использованием пакета рекомендуется задать драйвер по умолчанию. Сделать это можно в конфигурационном файле с расширением `cfg`. Например, для пакета `color` создается файл `color.cfg`, в первой строке которого указывается

```
\ExecuteOptions{dvipsone}
```

Теперь драйвером по умолчанию будет `dvipsone`. Перечень доступных для использованиея драйверов выглядит так:

```
dvips,xdvi,dvipdf,dvipdfm,dvipdfmx,pdftex,dvipsone,
```

```
dviwindo,emtex,dviwin,pctexps,pctexwin,pctexhp,
pctex32,truetex,tcidvi,vtex,oztex,textures,xetex
```

Существует возможность использовать и другие драйвера, не перечисленные в списке. Для этого в конфигурационный файл помещается строка с командой `\DeclareOption` с названием драйвера до вызова `\ExecuteOptions`. Например, чтобы добавить драйвер `dvi2ps` к списку драйверов, и сделать его драйвером по умолчанию, конфигурационный файл должен выглядеть примерно так

```
\DeclareOption{dvi2ps}{\def\Gin@driver{dvi2ps.def}}
\ExecuteOptions{dvi2ps}
```

Подходящий файл `dvi2ps.def` входит в дистрибутив, однако не доступен по умолчанию, поскольку он не был достаточно протестирован. Другие, входящие в дистрибутив, но не подключенные драйверы следующие

```
dvialw, dvilaser, dvitops, psprint, pubps, ln
```

Различные варианты дистрибутивов TEX поддерживают разные наборы драйверов, поскольку используют разные программы для печати. Если же требуется особенный драйвер, и его не удалось найти ни в одном дистрибутиве, то его можно попытаться создать самостоятельно, по аналогии с уже существующими.

1.2. Модель цвета

Поддержка цвета обычно основывается на понятии цветовой модели. Используемые модели могут изменяться в зависимости от конкретного драйвера, но обычно поддерживаются следующие:

- **RGB** (Red, Green, Blue). Тройка разделенных запятыми значений интенсивности компонент цвета. Каждая компонента задается числом от 0 до 1. Следует заметить, что в популярных массовых графических пакетах принято задавать значения интенсивности компонент цвета числами от 0 до 255 (или их шестнадцатеричными аналогами). Пере-

вод из одной системы в другую осуществляется простым умножением (или, соответственно делением) на 255.

- **CMYK** (Cyan, Magenta, Yellow, blacK). Четыре разделенных запятыми числа от 0 до 1, задающих компоненты цвета согласно субстрактивной модели CMYK, используемой многими устройствами печати.

- **Gray** (оттенки серого). Одно число он 0 до 1.

- **named**. Цвета задаются предопределенными именами, например `Red`. Эта модель поддерживается не всеми драйверами. Новые имена могут задаваться командами, описанными в документации. Некоторые из них будут упомянуты ниже.

Другие модели используются значительно реже, и не входят в стандартный дистриюутив, однако они могут быть описаны также, как и новые названия цветов в `named`.

1.3. Опции

Основная опция пакета — это указание на используемый драйвер. Другая опция, заслуживающая внимания, это опция `monochrome`. Она блокирует действие всех команд, связанных с изменением цвета. Ошибок они не вызывают, но и цвет не изменяют. Эта опция полезна при работе над цветным документом в среде, не поддерживающей отображение или печать цвета.

Остальные три опции пакета контролируют поведение модели `named`. Драйвер `dvips` (используемый по умолчанию) понимает 68 имен цветов. Поэтому использование драйвера `dvips` делает доступными эти имена для модели `named` сразу. Если же эти имена использоваться не будут, то их можно отключить опцией `nodvipsnames`. Если же, напротив, необходимо использовать имена драйвера `dvips` с другим драйвером, то этого можно достичь опцией `dvipsnames`. Наконец, опция `usenames` делает все имена модели `named` доступными напрямую, так, как описано ниже.

1.4. Определение цветов

Основные цвета определены по умолчанию, но список цветов можно расширять, пользуясь командой

```
\definecolor{}{<model>}{<colour specification>}
```

<name> это название нового цвета, <model> — используемая для задания цвета модель, и <colour specification> спецификация цвета в выбранной модели. Например,

```
\definecolor{light-blue}{rgb}{0.8,0.85,1}
\definecolor{mygrey}{gray}{0.75}
```

1.5. Использование цвета

Для смены цвета текста используется команда

```
\color{<name>}
```

Она изменяет текущий цвет на <name> до конца текущей группы или окружения. Другая команда

```
\textcolor{<name>}{<text>}
```

принимает текст, цвет каторого должен быть изменен, в качестве аргумента <text>. В некоторых случаях такой синтаксис удобнее, хотя по сути это эквивалентно конструкции {\color{<name>}text}.

Перечисленные команды используют оределенные заранее названия цветов <name>. Если по каким-то причинам нужно задать цвет явно, без предварительного именования, то сделать это можно соответственно командами

```
\color[<model>]{<specification>}
\textcolor[<model>]{<specification>}{<text>}
```

В частности, эти команды помогают использовать имена цветов модели named, например \color[named]{BrickRed}. Вместо того, чтобы все время использовать необязательный аргумент, можно создать алиас при помощи \definecolor, например

```
\definecolor{myred}{named}{WildStrawberry}
```

Если же имена модели named вполне соответствуют потребностям набора, то опция usenames пакета color автоматически создаст подобные алиасы для каждого из определенных в модели named цветов, что позволит использовать более короткую форму записи, например \color{Red} вместо \color[named]{Red}.

1.6. Цвет страницы

Фоновый цвет страницы можно установить командой

```
\pagecolor{<name>}
\pagecolor[<model>]{<specification>}
```

Значения параметров этих команд то же что и у команды \color. Они управляют цветом фона текущей страницы и всех последующих. Вернуть цвет фона в исходное состояние можно только явно, испульзуя команду \pagecolor{white}.

1.7. Цвет фона боксов

Две команды, похожие на \fbox, позволяют делать боксы, залитые выбранным цветом.

```
\colorbox{<name>}{<text>}
\colorbox[<model>]{<specification>}{<text>}
\fcolorbox{<name1>}{<name2>}{<text>}
\fcolorbox[<model>]{<specification1>}%
          {<specification2>}{<text>}
```

\colorbox создает бокс, окрашенный в цвет <name>, либо заданный спецификацией в определенной модели. Версия \fcolorbox создает бокс, залитый цветом <name2> с видимой рамкой вокруг, цвета <name1>. Эти команды используют параметры \fboxrule и \fboxsep для определения толщины линии и размера закрашиваемой области.

2. Пакет colortbl

Пакет colortbl предназначен для создания цветных таблиц, т.е. таблиц со цветными панелями позади содержимого ячейки. Для работы пакета требуется LaTeX и пакеты color и array.

2.1. Команда columncolor

Реализация команды в пакете такова, что она может использоваться только как аргумент спецификатора столбца > в преамбуле array или tabular, а также в \multicolumn. Формат команды следующий:

```
\columncolor[<color model>]{<colour>}%
    [<left overhang>][<right overhang>]
```

Аргументы <color model> и <colour> задают цвет подобно тому, как это происходит в команде \color, описанной выше. Аргумент <left overhang> и <right overhang> задают длину, на которую панель выступает за граница содержимого ячейки слева и справа соответственно. Если опущен один из размеров, то второй используется с обеих сторон. Если отсутствуют оба, то используется значение параметров \tabcolsep или \arraycolsep, в зависимости от окружения.

2.2. Команда rowcolor

Команда

```
\rowcolor[<color model>]{<colour>}%
    [<left overhang>][<right overhang>]
```

имеет такие же аргументы, как и \columncolor. Она должна использоваться в самом начале строки. Выступы определяются также аналогично \columncolor. Если команда используется в области действия \columncolor, то более высокий приоритет будет иметь \rowcolor. Команда \multicolumn может включать инструкцию >{\rowcolor..., которая будет более приоритетна, чем цвета, заданные для текуших столбца и строки.

Пример использования команды:

```
\begin{tabular}{|l|c|}
\rowcolor[gray]{.9}
содержимое 1.1&содержимое 1.2\\
\rowcolor[gray]{.6}
содержимое 2.1&содержимое 2.2
\end{tabular}
```

В результате получим:

содержимое 1.1	содержимое 1.2
содержимое 2.1	содержимое 2.2

2.3. Команда cellcolor

Можно задать цвет всей таблицы, или ее строк, но иногда бывает необходимо указывать цвет отдельной ячейки. Для этого служит команда

```
\cellcolor[<color model>]{<colour>}%
 [<left overhang>][<right overhang>]
```

которая имеет тот же синтаксис, и цвет, который она задает, преобладает над остальными установками. Использовать ее можно в любом месте ячейки, к которой она применяется.

2.4. Цвет границ ячеек

Можно задать цвет вертикальной границы, не используя никаких дополнительных команд, заменяя обычный разделитель столбцов `|` на `!{\color{green}\vline}`. Однако, раскрасить `\hline` и `\cline` не так просто, так что в пакет предоставляет в распоряжение авторов дополнительные команды.

```
\arrayrulecolor[<color model>]{<colour>}
```

имеет одинаковые параметры с `\color`, и задает цвет всех последующих горизонтальных и вертикальных границ во всех таблицах документа.

```
\doublerulesepcolor[<color model>]{<colour>}
```

работает подобно `\arrayrulecolor`, за исключением того, что промежутки, которые возникают в случае пустых ячеек типа `||` (или `\hline\hline`).

Следующего результата раскраски границ таблицы

содержимое 1.1	содержимое 1.2
содержимое 2.1	содержимое 2.2

можно добиться таким способом:

```
\setlength\arrayrulewidth{2pt}
\arrayrulecolor{blue}
\setlength\doublerulesep{2pt}
\doublerulesepcolor{yellow}
\begin{tabular}{||l||c||}
\hline\hline
содержимое 1.1&содержимое 1.2\\
содержимое 2.1&содержимое 2.2\\
\hline\hline
\end{tabular}
```

Лекция 17 ⏮

Подготовка и вставка плавающих объектов, обтекание текстом

Вставка таблиц и иллюстраций. Обтекание текстом. Использование пакетов wrapfig, floatflt и возможностей класса NCC

Эта лекция посвящена использованию в тексте дополнительных выразительных средств, таких как графические иллюстрации или таблицы. В терминах TEX они обозначаются как плавающие (`floating`) объекты.

1. Пакет floatflt

Пакет `floatflt` определяет два окружения: `floatingfigure` и `floatingtable`. Основная идея состоит в том, что если рисунок или таблица не очень широкие, то возле них может располагаться текст, справа или слева. Подключается пакет стандартным для LATEX образом

```
\usepackage[<option>]{<floatflt>}
```

где `<option>` может принимать одно из трех значений: `rflt`, `lflt`, или `vflt`. Эти опции устанавливает позицию для плавающего объекта по умолчанию:

- **rflt** — справа;
- **lflt** — слева;
- **vflt** — справа на нечетных и слева на четных страницах.

Последняя опция выбрана по умолчанию.

1.1. Окружение floatingfigure

Общий формат вызова окружения приведен ниже. Аргумент `<option>` может принимать значение `r`, `l`, `p`, или `v`, а `<width>` задает размер плавающего объекта.

```
\begin{floatingfigure}[<option>]{<width>}
вставка графического объекта
\caption %% если необходимо
\end{floatingfigure}
```

Опции обозначают следующее:

- **r** Объект будет выводиться справа.
- **l** Объект будет выводиться слева.
- **p** Объект будет выводиться справа на нечетных и слева на четных страницах.
- **r** Объект будет выводиться так, как определено опциями пакета, если же они не указаны, то справа на нечетных и слева на четных страницах.

1.2. Окружение floatingtable

Используется окружение следующим образом:

```
\begin{floatingtable}[<option>]{
\begin{tabular}{<tabular specifiers>}
содержание таблицы
\end{tabular}}
\caption %% если нужно
\end{floatingtable}
```

`<option>` может принимать одно из значений `r`, `l`, `p`, или `v`. Действие их аналогично окружению `floatingfigure`. Ширина явно не задается, зато окружение `tabular` целиком является обязательным аргументом.

2. Пакет wrapfig

Задачи, которые решает пакет `wrapfig`, практически те же, что и у пакета `floatflt`. Он определяет два окружения `wrapfigure` и `wraptable`, которые помещают рисунок или таблицу на одну из сторон страницы и располагают текст вокруг нее.

2.1. Окружение wrapfigure

Окружение имеет следующий формат вызова:

```
\begin{wrapfigure}[<number>] {<placement>}%
 [<overhang>] {<width>}
                    <figure>
\end{wrapfigure}
```

Параметр <number> задает количество строк текста для обтекания рисунка. Если параметр отсутствует, то количество определяется автоматически, в зависимости от высоты рисунка.

Параметр <placement> является обязательным, он определяет расположение рисунка на странице. Расположение может задаваться строчными R, L, I, O либо прописными r, l, i, o буквами. Строчные буквы задают желаемое расположение, т.е. предполагают возможность перемещения фигуры при необходимости. Прописные буквы указывают что фигура должна располагаться точно там где указано. Варианты расположения такие:

- **r, R** справа от текста
- **l, L** слева от текста
- **i, I** внутренний край страницы
- **o, O** внешний край страницы

Указать можно только одну из них.

Параметр <width> задает ширину рисунка. Значение этого параметра перекрывает вычисленное для рисунка значение ширины. Если значение параметра будет равно 0, тогда ширина будет определятся по ширине рисунка.

Необязательный параметр <overhang> определяет на сколько рисунок может перекрыть поля страницы. По умолчанию перекрытие задается параметром \wrapoverhang, который имеет значение 0. Изменить значение по умолчанию можно командой \setlength. Когда значение <overhang> явно задается для конкретного рисунка, можно использовать

специальный размер `\width`, который содержит фактическую ширину графического объекта.

2.2. Окружение wraptable

Окружение имеет следующий формат вызова:

```
\begin{wraptable}[<number>] %
      {<placement>}[<overhang>]{<width>}
   <table>
\end{wraptable}
```

Параметры окружения `wraptable` аналогичны только что описанным параметрам окружения `wrapfigure`.

3. Класс NCC

Класс NCC, разработанный А.И. Роженко, представляет собой масштабную доработку стандартных классов и пакетов LaTeX, и заслуживает отдельного обсуждения.

Команды, имеющие самостоятельное значение, вынесены в отдельные пакеты коллекции `ncctools`. Собственно макрос NCC и сопутствующие ему пакеты образуют коллекцию `ncclatex`. Ядро макроса интегрировано пакетом `ncclatex`, который можно использовать с другими классами.

Класс содержит множество полезных команд, которые существенно расширяют стандартные возможности LaTeX. Все они подробно описаны в документации http://www.ctan.org/get/macros/latex/contrib/ncclatex/ncclatex.pdf, том числе и на русском языке http://www.ctan.org/get/macros/latex/contrib/ncclatex/nccnews.pdf.

В частности, отдельно описаны команды, позволяющие оформлять с обтеканием или в виде плавающих объектов рисунки и таблицы.

3.1. Основные команды

Следующая команда задает стиль `<style>` для оформления плавающих объектов (иллюстраций и таблиц):

```
\FloatingStyle{<style>}
```

По умолчанию заден стиль

```
\FloatingStyle{\footnotesize\centering}
```

Для размещения рисунков и таблиц в виде мини-сираницы служат команды

```
\minifig[<pos>][<высота>][<innerpos>]{<ширина>}{<тело>}
\minitabl[<pos>][<высота>][]{<ширина>}{<тело>}
```

Параметр `<pos>` определяет как объект будет выравниваться по вертикали. Он может принимать одно из трех значений:

- **t** выравнивание по верхнему краю
- **t** выравнивание по нижнему краю
- **c** выравнивание по центру

Параметры `<высота>` и `<ширина>` задают высоту и ширину соответственно, `<innerpos>` позволяет указать выравнивание внутри мини-страницы (допустимые значения (`t`, `b`, `c` или `s`). Параметр `<тело>` содержит команды, непосредственно вставляющие требуемый объект, в том числе и команду `\caption` при необходимости. На этих базовых командых основаны все остальные команды пакета, связанные с оформлением плавающих объектов.

3.2. Обтекание текстом

Пакет `NCC` отличается от описанных выше тем, что в нем реализованы средства ручного контроля за обтеканием. Для размещения объекта в тексте с обтеканием используется одна из команды:

```
\sidefig[<pos>](<w1>)(<w2>){<рисунок>}{<текст>}
\sidefig*[<pos>](<w1>)(<w2>){<рисунок>}{<текст>}
\sidetabl[<pos>](<w1>)(<w2>){<таблица>}{<текст>}
\sidetabl*[<pos>](<w1>)(<w2>){<таблица>}{<текст>}
```

По умолчанию рисунок или таблица располагается на четных страницах слева и на нечетных справа. Версии команд со звездочкой размещают объ-

ект на противоположной стороне. Необязательные параметры имеют следующие значения:

- **pos** определяет вертикальное выравнивание относительно текста (t, b или c)
- **w1** задает ширину плавающего объекта в единицах `\unitlength`
- **w2** задает ширину текстового бокса.

3.3. Оформление рисунка или таблицы

Следующие команды фактически являются надстройками над окружениями `figure` и `table`:

```
\fig[<placement>](<w>){<рисунок>}
\fig*[<placement>](<w>){<рисунок>}
\tabl[<placement>](<w>){<таблица>}
\tabl*[<placement>](<w>){<таблица>}
```

`<placement>` задает предпочтительные варианты размещения на странице, `<w>` задет ширину.

Следующие команды размещают рядом пару плавающих объектов:

```
\figs[<placement>](<w1>)(<w2>)%
                  {<рисунок 1>}{<рисунок 2>}
\figs*[<placement>](<w1>)(<w2>)%
                  {<рисунок 1>}{<рисунок 2>}
\tabls[<placement>](<w1>)(<w2>)%
                  {<таблица 1>}{<таблица 2>}
\tabls*[<placement>](<w1>)(<w2>)%
                  {<таблица 1>}{<таблица 2>}
```

Параметр `<placement>` имеет то же значение, что и для предыдущих команд, параметры `<w1>` и `<w2>` задают ширину левого и правого элемента соответственно.

Лекция 18 ⏮

Создание документов в формате PDF. Проект PDFTEX

Основные понятия и команды. Графика и цвет.

Главная задача проекта PDFTEX состоит в том, чтобы

- из исходного файла TEX можно было создавать документ в формате PDF;
- расширить возможности TEX за счет использования возможностей формата PDF, например создавать активные гиперссылки.

PDFTEX входит в состав большинства современных дистрибутивов TEX, в ином случае он может быть установлен дополнительно. Получить его можно по адресу ftp://ftp.cstug.cz/pub/tex/local/cstug/thanh/pdftex/latest. Когда все установлено, использовать новые возможности можно просто вызывая для обработки исходного TEX-файла программу pdftex вместо tex (или pdflatex вместо latex), например

```
# pdftex myfile.tex
```

Заметим, что если отключить вывод в формат PDF, то PDFTEX создаст файл в привычном формате DVI.

Все возможности PDFTEX подробно описаны в документации. Далее мы познакомимся с некоторыми из них.

1. Информация о документе

Просматривая документ в формате PDF программой Adobe Acrobat Reader, получить информацию о документе можно нажав сочетание клавиш Ctrl-D (или выбрав в меню Файл пункт Свойства документа). Добавить такую информацию в свой файл PDF можно командой

```
\pdfinfo{<info>}
```

где <info> задается в стиле PDF, в виде пар

```
/fieldname (<text>)
```

Возможные значения для `/fieldname` такие:

```
/Title
/Author
/Creator
/Producer
/CreationDate
/ModDate
/Subject
/Keywords
```

Дата задается в формате `D:YYYYMMDDHHmmss`. Пакет `datetime` (начиная с версии 2.31) определяет команду `\pdfdate`, которая вставляет дату в правильном формате. Приведем пример:

```
\pdfinfo{
/Author (The Author)
/Title (Creating PDF using PDFTeX)
/CreationDate (D:200812312359)
/ModDate (D:\pdfdate)
/Subject (PDFTeX)
/Keywords (PDF;TeX)
}
```

Используя команды PDFTEX необходимо помнить, что для обычного TEX (или LATEX) эти команды неизвестны, и вызовут ошибку. Для того, чтобы сделать исходный файл универсальным, необходимо иметь возможность проверить, какой программой он обрабатывается. Это можно сделать, подключив пакет `ifpdf`, в котором определена команда-условие `\ifpdf`, позволяющая проверить, используется PDFTEX или нет. Например, команду `\pdfinfo` рекомендуется использовать следующим образом:

```
\ifpdf
\pdfinfo{
/Author (The Author)
/Title (Creating PDF using PDFTeX)
/CreationDate (D:200812312359)
/ModDate (D:\pdfdate)
/Subject (PDFTeX)
/Keywords (PDF;TeX)
}
\fi
```

2. Включение графики в PDF-документы

Для включения в документы PDF графики можно использовать пакеты graphics (graphicx), которые будут описаны позднее. Здесь только отметим, что, к сожалению, придется отказаться от использования графических файлов в формате PS или EPS, поскольку PDFTEX не умеет преобразовывать такие форматы. Ему нужны либо файлы в формате PDF, либо в одном из поддерживаемых графических форматов, например PNG или TIFF.

Хорошей новостью можно считать то, что использовать специальные средства (такие как \ifpdf) для подгрузки требуемого файла в зависимости от выходного формата (PDF или DVI) не потребуется. Дело в том, что команда включения графики \includegraphics может автоматически определить расширение файла, который требуется в каждом конкретном случае. Таким образом достаточно иметь два графических файла, один в формате EPS с расширением .eps, а другой, например, в формате TIFF с расширением .tif, и указать в команде \includegraphics только имя файла, не указывая расширения. Например,

```
\includegraphics{shapes}
```

Тогда, в случае если файл будет обрабатываться системой PDFLATEX, будет использован файл shapes.tif, в противном случае подгрузится shapes.eps. Достигается это засчет способности графических пакетов

автоматически определять допустимые расширения файлов, как будет рассказано ниже.

Вместо пакета PSTricks для создания рисунков прямо внутри документа можно использовать аналогичный по набору функций пакет PDFTricks.

3. Гиперссылки. Пакет hyperref

Включение гиперссылок в PDF-документы осуществляется при помощи пакета hyperref. Для детального знакомства с этим пакетом необходимл воспользоваться документацией к нему. Здесь будут приведены только основные команды и примеры их использования.

3.1. Перекрестные ссылки и цитаты

Как только пакет hyperref будет подключен командой \usepackage, все перекресные ссылки и цитаты, созданные командами \ref, \pageref и \cite, автоматически будут сделаны активными гиперссылками. По умолчанию в документе они выглядят как цветные (красные) рамки вокруг текста ссылки. Это поведение может быть изменено одной из опций пакета. Например, если указана опция colorlinks, то активные ссылки будут просто выделяться цветом.

Пакет hyperref определяет также версии команд \ref* и \pageref*, которые печатают соответствующий номер, однако не создают гиперссылки, что позволяет их использовать с другими командами.

Следующий способ создать гиперссылку, это использовать команду

```
\hyperref[<label>]{<text>}
```

<text> выделяется цветом и становится активной ссылкой на объект, имеющий метку <label>.

Ссылка на URL создается при помощи команды

```
\href{<url>}{<text>}
```

<text> выделяется цветом и становится гиперссылкой на <url>. Например,

```
\href{http:\\www.sfu-kras.ru}%
{Сибирский федеральный университет.}
```

В адресе `<url>` могут быть напрямую использованы символы ~ и #. Если в качестве текста должен быть использован сам адрес, то можно использовать команду

```
\url{<address>}
```

которая эквивалентна `\href{<address>}{<address>}`.

3.2. Содержание и закладки

Содержание, производящееся командой `ntableofcontents`, автоматически будет состоять из активных ссылок на соответствующие разделы. По умолчанию ссылками будут сделаны заголовки разделов, а не номера страниц, однако обратить ситуацию можно использовав опцию `linktocpage` пакета `hyperref`.

Если установлена опция `bookmarks`, то будет создан набор закладок документа PDF, доступ к которым может быть получен выбором опции `Закладки` в меню `Окно` программы `Acrobat Reader`. Закладки являются удобным средством навигации по документу. Если также будет использована опция `bookmarksopen`, то по умолчанию дерево закладок будет полностью развернуто. Опция `bookmarksnumbered` добавит к тексту закладки номер раздела, на который она ссылается. Необходимо помнить, что в закладках нельзя использовать никакое форматирование LaTeX, в том числе и математический режим.

3.3. Разметка страницы

Размер страницы устанавливается одной из опций

```
a4paper, a5paper, b5paper,
letterpaper, legalpaper, executivepaper.
```

Также можно использовать опции, которые изменят режим показа документа, вот некоторые из них:

Опция	*Назначение*
`pdfcenterwindow`	Разместить окно документа по центру
`pdffitwindow`	Подогнать размер окна под размер первой страницы документа
`pdftoolbar`	Сделать панель инструментов видимой (`=true`) или невидимой (`=false`)
`pdfmenubar`	сделать строку меню видимой (`=true`) или невидимой (`=false`)
`pdfpagelayout`	Установить разметку страницы при открытии документа. Возможные значения: `SinglePage`, `OneColumn`, `TwoColumnLeft`, `TwoColumnRight`

3.4. Доступ к меню Acrobat Reader

Следующая команда используется, чтобы получить из документа доступ к меню и функциям `Acrobat Reader`:

```
\Acrobatmenu{<menuitem>}{<text>}
```

Эта команда создает в документе активную ссылку с текстом `<text>`, которая вызывает команду меню `<menuitem>`. В качестве текста может быть использована любая корректная конструкция LaTeX, в том числе и рисунки. Полный список возможных значений `<menuitem>` приводится в документации, наиболее востребованные из них `PrevPage`, `NextPage`, `FirstPage`, `LastPage`, `GoBack`, `GoForward` и `Quit`.

Лекция 19 ⏮

Подготовка качественных иллюстраций для LaTeX и их вставка

основные форматы графических файлов и возможности их включения в документ, подготовленный в LaTeX. Пакет graphicx и декларации для включения графических файлов в документ. Манипулирование графическими объектами.

1. Пакеты graphics и grapicx

В дистрибутиве LaTeX поставляются два пакета для работы с графикой: стандартный пакет для работы с графикой `graphics` и расширенная версия этого пакета `graphicx`. Отличаются они только форматом необязательных параметров команд, обязательные параметры одинаковые для обоих пакетов.

1.1. Опции пакетов

Графические пакеты, как и пакеты управления цветом, зависят от используемого графического драйвера. Использовать разные драйверы в одном документе невозможно, поэтому выбрав драйвер в однгом из пакетов указывать его в другом уже не нужно. Глобальные установки для графических пакетов хранятся в файле `graphics.cfg`. Чтобы использовать драйвер `dvips`, укажите в первой строке этого файла

```
\ExecuteOptions{dvips}
```

Кроме опции драйвера, графические пакеты имеют несколько опций, которые управляют набором предоставляемых возможностей.

- **draft** минимизирует набор используемых специальных возможностей. В частности, рисунки не отображаются, вместо них показывается рамка, соответствующая размеру рисунка и внутри нее текст с названием графического файла с рисунком.

- **final** противоположна `draft`. Можно использовать для отмены действия глобальной опции `draft`, указанной при загрузке класса.
- **hiderotate** Отменяет показ повернутого текста.
- **hidescale** Отменяет показ масштабированного текста.
- **hiresbb** Указывает брать размеры из строки `%%HiResBoundingBox`, а не из стандартного `%%BoundingBox`.
- **demo** команда `\includegraphics` вместо включения графического файла рисует прямоугольник размерами 150 на 100 пунктов, если только другие размеры не заданы вручную.

1.2. Вращения

```
graphics: \rotatebox{<angle>}{<text>}
graphicx: \rotatebox[<key val list>]{<angle>}{<text>}
```

Эти команды создают бокс с текстом `<text>` и поворачивают его на `<angle>` градусов. Стандартная версия всегда поворачивает бокс вокруг опорной точки, а расширенная позволяет изменять это поведение посредством параметров `<key val list>`, которые могут быть следующие

```
origin=<label> x=<dimen> y=<dimen> units=<number>
```

Так что можно задать координаты центра вращения `x` и `y`, либо указать одну из предопределенных точек, задав соответствующий `origin`. Например, `[origin = c]` задает вращение вокруг центра бокса, а `[origin = tr]` — вокруг верхнего правого угла. Параметр `units` позволяет изменить единицы измерения угла вращения, например `[units = -360]` задает вращение против часовой стрелки, а `[units= 6.283185]` позволяет указывать углы в радианах.

1.3. Масштабирование

Графические пакеты поддерживают различные способы масштабирования текста.

```
\scalebox{<h-scale>}[<v-scale>]{<text>}
```

задает масштабные множители по горизонтали (`<h-scale>`) и вертикали (`<v-scale>`). Если на задан параметр `<v-scale>`, то для растяжения по вертикали используется `<h-scale>`. Команда

```
\reflectbox{<text>}
```

является аббревиатурой (макроопределением) для `\scalebox{-1}[1]{<text>}`.

Команда

```
\resizebox*{<h-length>}{<v-length>}{<text>}
```

масштабирует `<text>` таким образом, чтобы он стал шириной <h-length>. Если в качестве одного из инмерений указан `!`, то команда будет подбирать по этому измерению такой множитель, чтобы масштабирование по всем направлениям было одинаковым. Обычно размер `<v-length>` относится именно к высоте бокса, но в версии этой команды со звездочкой он обозначает высоту+глубину бокса.

1.4. Вставка графических файлов

Функции по включению графики устроены максимально универсальным образом, для того чтобы включать графические файлы любого из поддерживаемых драйвером типов. Они опираются на перечень расширений файлов, которые конкретный драйвер может обрабатывать, хотя этот перечень может быть расширен.

Если расширение файла не известно драйверу, система пытается отобразить его, предполагая что файл имеет тип по умолчанию. Например, драйвер `dvips` считает типом по умолчанию тип `eps` (`PostScript`).

Графические файлы включаются в документ при помощи двух команд: `graphics` и `graphicx`.

```
graphics:
   \includegraphics*[<llx,lly>][<urx,ury>]{<file>}
graphicx:\includegraphics*[<key val list>]{<file>}
```

Если в команде присутствует символ $*$, то рисунок будет обрезан до указанного в необязательном аргументе размера, в противном случае, часть рисунка, выходящая за пределы указанных размеров, будет накладываться на текст. Если необязательные аргументы, задающие размер, отсутствуют, то информация о размере берется из самого графического файла в соответствии с перечисленными ниже правилами.

- **graphics**ли параметр `[<urx,ury>]` присутствует, то он указывает на координаты правого верхнего угла рисунка, в терминах размеров TEX. Если не указаны единицы измерения, то по умолчанию используется `bp`. Если присутствует параметр `[<llx,lly>]`, то он указывает координаты левого нижнего угла рисунка, иначе эти координаты принимаются равными `[0,0]`.

- **graphicx**рсия команды со звездочкой в этом пакете оставлена только для обратной совместимости. Она просто добавляет ключ `clip` к списку ключей `<key val list>`. Также для совместимости в случае, когда указаны два необязательных аругмента, будет использоваться стандартная версия команды, даже если загружен расширенный пакет.

Список допустимых значений для `<key val list>` выглядит так:

- **bb** В качестве аргумента указываются четыре измерения, разделенные пробелами. Это границы области, в которой будет отображен рисунок (Bounding Box)

- **bbllx,bblly,bburx,bbury** устанавливают размеры области отображения. Используются преимущественно для совместимости. `bbllx=a,bblly=b,bburx=c,bbury=d` эквивалентно `bb=abcd`.

- **natwidth,natheight** Еще одна альтернатива `bb`, `natheight=h,natwidth=w` эквивалентно `bb = 0 0 h w`.

- **hiresbb** Булевский ключ, если установлен в значение истина (его появление в списке устанавливает его в значение истина), то TEX

будет ориентироваться на строки `%%HiResBoundingBox`, а не на `%%BoundingBox`. Может быть явно установлен в значение ложь (`hiresbb=false`), чтобы перекрыть действие какой-либо глобальной опции, устанавливающей его в значение истина.

- **viewport** принимает, подобно `bb`, четыре аргумента, однако они будут рассматриваться как координаты, заданные относительно левого нижнего угла области вывода.
- **trim** Также имеет четыре аргумента, которые определяют, сколько нужно добавить или сколько отнять от каждой из сторон области вывода.
- **angle** Угол поворота.
- **origin** Опорная точка для поворота.
- **width** Требуемая ширина. Рисунок будет масштабирован до этой ширины.
- **height** Требуемая высота. Рисунок будет масштабирован до этой высоты.
- **totalheight** Полная высота рисунка (высота+глубина).
- **keepaspectratio** Булевский ключ (принимает значения истина или ложь). Если истина, то при указании и высоты и ширины не будут нарушены пропорции изображения.
- **scale** Масштабный множитель.
- **clip** Истина или ложь (если значение на задано явно, то истина). Графика будет обрезана по границе области вывода.
- **draft** Булевский ключ, локально переключает отображение именно этого рисунка в черновой режим (отображается только рамка и название файла).

- **type** Тип графики.
- **ext** Расширение файла. Может быть использован только вместе с `type`.
- **command** Указывает команду, которая должна выполняться для загружаемого файла. Может быть использован только вместе с `type`.

Первые семь ключей задают размеры картинки. Размер требуется указывать в том случае, когда файл не может быть прочитан или содержит неверные данные о размере.

Несколько последующих ключей управляют масштабом и вращением рисунка. Чтобы добиться того же эффекта в стандартном пакете, необходимо использовать команду `\includegraphics` в качестве аргумента команды `\rotatebox` или `\scalebox`.

2. Ряд команд пакета graphics

```
\graphicspath{<dir-list>}
```

Эта команда используется для задания списка каталогов для поиска в них графических файлов. Каждая директория в списке должна быть окружена парой фигурных скобок, даже если она в списке одна. Например,

```
\graphicspath{{eps/}{tiff/}}
```

дает указание искать графические файлы в поддиректориях `eps` и `tiff` текущей директории. По умолчанию, TEX будет искать графические файлы там же, где он ищет и остальные. То есть в директориях, заданных

```
\DeclareGraphicsExtensions{<ext-list>}
```

Эта команда определяет поведение TEX в случае, когда не указано расширение графического файла. <ext-list> содержит разделенные запятыми расширения.

Отметим, что если при включении графического файла не указано его расширение, то файл обязан существовать, иначе возникнет сообщение об ошибке. Если же расширение указано явно в команде

\includegraphics, то в процесе работы LaTeX файла может и не быть, но тогда в аргументах должны явно указываться размеры области вывода.

```
\DeclareGraphicsRule{<ext>}{<type>}{}{<command>}
```

Определяет правила обработки файлов с расширением <ext>. Может встречаться несколько раз (по разу для каждого из расширений). Допускается использовать в качестве расширения символ *. В этом случае правило будет применяться ко всем расширениям, для которых правила не заданы явно.

<type> указывает тип графического файла, для файлов с разными расширениями могут иметь один тип, например расширения ps, eps, ps.gz соответствует типу файла eps.

<read-file> указывает расширение файла, содержащего информацию о размерах изображения. В большинстве случаев размеры хранятся в том же файле что и сама графика, однако в некоторых случаях прочитать размеры из графического файла не представляется возможным (например, он не существует). Если этот аргумент оставлен пустым, то система не будет пытаться взять размеры из какого-либо файла, и в этом случае они должны быть явно указаны в аргументах \includegraphics. Наконец, в качестве этого аргумента может быть указан символ *, и тогда расширение файла с размерами будет такое же, как и файла с графикой. Правило по умолчанию выглядит так: \DeclareGraphicsRule{*}{eps}{*}{}, т.е. все неизвестные расширения будут рассматриваться как файлы типа EPS, и размеры изображения будут браться из самого файла.

Аргумент <command> обычно остается пустым, однако если он не пуст, то он должен указывать на имя исполняемого файла, который будет запущен перед загрузкой графики. Внутри этого аргумента может использоваться #1, для обозначения имени открываемого файла. Например,

```
\DeclareGraphicsRule{.ps.gz}{eps}{.ps.bb}{`zcat #1}
```

будет вызывать архиватор для распаковки упакованных графических файлов.

Лекция 20 ⏮

Гипертекст в PDF

Пакет hyperref и его опции. Навигация на базе перекрестных ссылок. Гиперссылки внутри документа, на другой документ, на ресурсы в сети. Обратные гиперссылки из библиографического списка. Запуск внешнего приложения.

Пакет `hyperref` расширяет функциональные возможности LaTeX'а, делая перекрестные ссылки на команды оглавления, библиографии и т.д. Он также обеспечивает новые команды для ссылок к внешним документам.

Загружаем пакет `hyperref`:

```
\usepackage[<opt>]{hyperref}
```

При загрузке этого пакета после компиляции документа на выходе по умолчанию появится закладка рядом с текстом (слева), на которой выведется список разделов документа. Если закладка будет не нужна, её можно убрать, написав опцию `bookmarks=false` (о которой говорится дальше) при загрузке пакета `hyperref`.

Главы в содержании будут обведены рамочкой красного цвета (цвет шрифта, математики и др. не изменится), а также появится ссылка из содержания к странице, на которой начинается соответствующий раздел документа. Все ссылки внутри документа будут обведены красной рамочкой, ссылки на другой файл обведены темно-зеленой рамочкой, ссылки на сайт (интернет) — голубой рамочкой, ссылки на библиографию — салатовой рамочкой. При нажатии на ссылку, она станет в боксе черного цвета, сама ссылка станет желтого цвета.

Включая опции `<opt>`, описанные ниже, можно изменять цвета ссылок и внешний вид документа.

1. Команды, определенные в пакете hyperref

Пакет `hyperref` предназначен в основном для гипертекстовых ссылок. Три описанные ниже команды устанавливают эти ссылки.

```
\href{file}{text}
```

Ссылка на файл `file.pdf` (указывается имя `file` файла `file.pdf` без расширения). С помощью этой команды можно делать ссылку и на сайт в интернете. Для этого вместо `file` нужно написать адрес сайта в интернете, например, написав

```
\href{http://www.sfu-kras.ru}{ссылку на сайт СФУ}
```

получим: ссылку на сайт СФУ.

```
\hypertarget{name}{text}
```

Эта команда подобна команде `\label` из LaTeX'а, она ставит видимую метку `text`. В `text` можно написать любое слово, знак или символ, а можно вообще ничего не писать (оставить пустые скобки).

```
\hyperlink{name}{text}
```

Эта команда возвращает к метке, установленной с помощью команды `\hypertarget`. Здесь текст `text` должен быть обязательно написан, это может быть также слово, знак или символ. Имена `name` должны быть одинаковы, а вот текст `text` — не обязательно. Если метка `\hyperlink` дана, а соответствующей метки `\hypertarget` нет, то ссылка принесет к началу документа, т. к. не будет знать куда ссылаться.

Например, поставим в начало пункта «Закладки» данной лекции метку, написав `\hypertarget{sec1}{}`, а затем в тексте документа, например, прямо здесь поставим вторую метку `\hyperlink{sec1}{к закладкам}`: к закладкам. Теперь если нажать на текст «к закладкам», то мы попадем к тому месту, где стоит первая метка, т.е. к началу пункта «Закладки».

Кроме этих ссылок также работают ссылки из LaTeX'а: `\label`, `\ref`, `\pageref`, `\cite`.

2. Опции пакета

Пакет `hyperref` имеет большое количество опций. Опции перечисляются через запятую в необязательном аргументе команды `\usepackage` при подключении пакета

```
\usepackage[список опций]{hyperref}
```

либо после подключения пакета в аргументе команды

```
\hypersetup{список опций}.
```

Команда `\hypersetup` используется не только в преамбуле, но и в окружении `document` для того, чтобы менять различные `PDF` настройки прямо по ходу документа. Каждая опция в списке задается в формате

```
опция=значение
```

Для опций булевого типа значение `true` можно опускать.

Мы рассмотрим только основные опции. Более полный список можно найти в документации к пакету.

2.1. Внешний вид гиперссылок

Оформление текста гиперссылки регулирует двt опции булевого типа:

```
colorlinks,    frenchlinks
```

По умолчанию значения обеих опций равно `false` и текст гиперссылки печатается обычным шрифтом без выделения цветом, но вокруг него рисуется рамка. Если `colorlinks` присвоить значение `true`, то текст гиперссылки будет выделяться цветом, а рамки не будет.

Если же `frenchlinks` присвоить значение `true`, то текст гиперссылок будет печататься шрифтом КАПИТЕЛЬ без выделения цветом. При вычислении координат углов прямоугольника, который становится активной областью гиперссылки на `PDF` странице, LaTeX добавляет поля, величина которых задается опцией

```
pdflinkmargin=длина
```

Вместе с величиной указываются, как это принято в LaTeX'е, и единицы изменения, например, 10pt. По умолчанию действует значение 1pt. Опцию имеет смысл использовать для увеличения расстояния между рамкой и текстом ссылки.

2.2. Цвет текста гиперссылок

Следующие опции определяют цвет текста ссылок разного типа:

```
linkcolor pagecolor filecolor  citecolor  urlcolor
```

Цвет текста ссылок на мишени внутри документа определяет `linkcolor`; по умолчанию имеет значение red. Цвет текста ссылок, когда мишенью внутри документа является страница, определяет `pagecolor`; по умолчанию имеет значение red. Цвет текста ссылок, которые открывают локальные `PDF` файлы задает `filecolor`; по умолчанию имеет значение cyan. Цвет библиографических ссылок, которые печатает команда `\cite`, определяет `citecolor`; по умолчанию имеет значение green. Цвет текста ссылок на ресурсы с `URL` определяет `urlcolor`; по умолчанию имеет значение magenta. Допустимыми значениями перечисленных выше опций могут быть имена цветов из пакета `color` или имена, определенные посредством команды `\definecolor` из того же пакета, причем команда становится доступной только после подключения самого пакета. Опция

```
anchorcolor=цвет
```

задает цвет текста мишени; по умолчанию **black**.

2.3. Окантовка гиперссылок

По умолчанию вокруг гиперссылок рисуется прямоугольная рамка. Линия рамки имеет толщину 1 bp. Опция

```
pdfborder={стиль рамки}
```

позволяет изменить вид рамки. Стиль рамки в общем случае состоит из трех целых чисел и одного массива чисел. Первые два числа — радиусы закругления углов рамки, а третье число - толщина линии рамки. Если в качестве толщины линии указать 0, то рамки не будет. Необязательный массив позволяет заменить сплошную линию на штриховую. Первое число в массиве задает длину штриха, а второе — расстояние между штрихами (не должны превышать 10). Пример: `pdfborder={0 0 2 [6 4]}`. Если расстояние между штрихами совпадает с длиной штриха, то вместо [n n] можно указывать [n]. По умолчанию стиль рамки задан как 0 0 1.

Следующие опции определяют цвет рамки вокруг ссылок разного типа:

```
linkbordercolor
```

Цвет рамки для гиперссылок на мишени внутри документа; по умолчанию имеет значение 1 0 0. Пример: `linkbordercolor=1 0 0`.

`pagebordercolor`

Цвет рамки для гиперссылок, когда мишенью внутри документа является страница; по умолчанию имеет значение 1 1 0.

`filebordercolor`

Цвет рамки для гиперссылок, которые открывают локальные PDF файлы; по умолчанию имеет значение 0 .5 .5.

`citebordercolor`

Цвет рамки для библиографических ссылок, которые печатает команда `\cite`; по умолчанию имеет значение 0 1 0.

`urlbordercolor`

Цвет рамки для гиперссылок на ресурсы с URL; по умолчанию имеет значение 0 1 1.

`runbordercolor`

Цвет рамки для гиперссылок по схеме run; по умолчанию имеет значение 0 .7 .7. Цвет в перечисленных выше опциях задается в цветовой модели RGB как три числа от 0 до 1.

2.4. Выделение активных гиперссылок

`pdfhighlight=ключ`

— опция устанавливает выделение гиперссылок, когда кнопка мыши нажимается и держится нажатой, а указатель мыши находится в активной области. Допустимые значения опции: ключ `/N` — не выделяется; `/I` — инвертируется цвет содержания; `/O` — инвертируется цвет границы; `/P` — имитируется вдавливание (неплохо смотрится на цветном или сером фоне). По умолчанию действует ключ `/I`. Следующие ссылки демонстрируют действие ключа `/I`, ключа `/O`, ключа `/P`.

2.5. Разрыв текста гиперссылки

Опция булевого типа

```
breaklinks
```

разрешает/запрещает разбивать текст гиперссылки для переноса на другую строку. Для драйвера `pdftex` значение по умолчанию равно `true`, в остальных случаях — `false`.

2.6. Закладки

Опция булевого типа

```
bookmarks
```

определяет, будут или не будут созданы закладки для документа. По умолчанию опция имеет значение true. Для того, **чтобы в закладках были видны русские буквы**, пакет **hyperref** надо подключать с опцией **unicode**. Вид закладок в Acrobat Reader регулируется следующими опциями:

```
bookmarksnumbered
```

задает нумерацию разделов; по умолчанию — false.

```
bookmarksopen
```

раскрывает полностью дерево закладок; по умолчанию — false.

bookmarksopenlevel=целое число

Задает уровень, до которого раскрывается дерево закладок; можно менять по ходу документа. По умолчанию раскрывается полностью.

```
bookmarkstype
```

устанавливает, какой файл с таблицей содержания воспроизводится в закладках; по умолчанию — `toc`.

2.7. Опции, влияющие на создание гиперссылок

По умолчанию гипертекстовыми ссылками является текст элементов оглавления и списков таблиц и рисунков. Опция булевого типа

```
linktocpage
```

делает гиперссылками номера страниц в этих таблицах, а не текст элементов. Значение по умолчанию — `false`.

Опция булевого типа

```
hyperfootnotes
```

определяет, будут или не будут подстрочные примечания гипертекстовыми ссылками. Значение по умолчанию — `true`.

Опция булевого типа

```
hyperindex
```

определяет, будут или не будут номера страниц в предметном указателе гипертекстовыми ссылками. Значение по умолчанию — `true`. Если в команде спользуются команды выбора шрифта для печати номера станицы, которые, как известно, начинаются в этом случае с символа |, то такие номера не становятся гиперссылками. Дело в том, что пакет hyperref сам использует в `\indexentry` символ | для записи команды `\hyperpage` Проблема решается очень просто. Надо ввести новую команду, например, `\Emph` в виде и вместо `\index{термин|emph}` писать `\index{термин|Emph}`.

Опция булевого типа

```
pageanchor
```

определяет, будет или не будет создана мишень в левом верхнем углу на каждой странице. Значение по умолчанию — `true`. Если опцию отключить, то номера страниц в предметном указателе не будут работать как гипертекстовые ссылки, поскольку они используют эти мишени.

Пакет `hyperref` позволяет создать обратные гиперссылки из библиографического списка: в конец каждого элемента списка добавляются в виде гиперссылок номера страниц или номера разделов, где встречаются прямые ссылки на этот элемент. Обратные гиперссылки в виде номера страницы создаются, если задана опция

```
pagebackref
```

По умолчанию ее значение равно `false`. При наличии опции

```
backref
```

(по умолчанию ее значение равно `false`) создаются гиперссылки в виде номера разделов, причем только в том случае, когда после каждого элемента библиографического списка оставлена пустая строка или стоит команда.

Опцией булевого типа

```
draft
```

можно отключить все гипертекстовые опции. По умолчанию – `false`.

2.8. Презентации

Опции `pdfpagetransition` и `pdfpageduration` определяют, как страницы PDF документа сменяют друг друга на экране, и используются обычно при создании электронных презентаций. В `Acrobar Reader` в полноэкранной моде просмотра документа можно установить разные типы появления на экране новых страниц. Пакет `hyperref` позволяет через опцию

```
pdfpagetransition=тип перехода
```

задать любой тип, причем менять тип перехода можно по ходу документа, задавая его в аргументе команды `\hypersetup.`

`Тип перехода` состоит из названия перехода и набора необязательных ключей для выбора разных вариантов одного типа. Набор ключей зависит от типа перехода. Пример: `\hypersetup`pdfpagetransition=Box/D2/M/O

Ниже описаны все типы переходов, включая их названия и ключи:

```
Split
```

Две линии, скользя по экрану, открывают новую страницу. Линии могут быть горизонтальными или вертикальными. Они движутся либо навстречу друг другу с краев экрана к центру, либо от центра к краям. Тип линий задается ключом `/Dm` с допустимыми значениями `/H` (горизонтальные) и `/V` (вертикальные). Направление движения регулируется ключом `/M`, который может принимать значение `/I` (внутрь) или `/O` (наружу). По умолчанию действуют ключи: `/Dm /H /M /I`.

```
Blinds
```

Несколько эквидистантных линий, скользя синхронно по экрану, открывают новую страницу. Линии могут быть горизонтальными или вертикальными. Их тип задается ключом `/Dm` с допустимыми значениями `/H` (горизонтальные) и `/V` (вертикальные). Горизонтальные линии движутся вниз, а вертикальные — вправо. По умолчанию действует ключ: `/Dm /H`.

`Box`

Квадратная рамка, сжимаясь с краев к центру экрана или разжимаясь из центра экрана, открывает новую страницу. Направление движения регулируется ключом `/M`, который может принимать значение `/I` (внутрь) или `/O` (наружу). По умолчанию действует ключ: `/M /I`.

`Wipe`

Прямая линия, скользя от одного края экрана к противоположному, открывает новую страницу. Направление движения регулируется ключом `/Di`, который может принимать значение 0 (слева направо), 90 (снизу вверх), 180 (справа налево) или 270 (сверху вниз). По умолчанию действует ключ: `/Di` 0.

`Dissolve`

Старая страница исчезает маленькими кусочкам.

`Glitter`

Старая страница исчезает маленькими кусочкам, но в отличие от `Dissolve`, не на всей странице сразу, а в широкой полосе, которая скользит по экрану. Направление движения регулируется ключом `/Di`, который может принимать значение 0 (слева направо), 270 (сверху вниз) или 315 (из левого верхнего угла в правый нижний). По умолчанию действует ключ: `/Di` 0.

`R`

Новая страница замещает старую без каких-либо эффектов. Все типы перехода, кроме `R`, имеют также опцию `/D`. Она задает длительность перехода в секундах. По умолчанию значение опции равно 1. Опция

`pdfpageduration=число`

позволяет задать автоматический режим смены страниц. В качестве значения опции указывается время в секундах, в течение которого текущая страница отображается на экране перед тем, как будет выведена следующая. Пользователь может в любое время сам сменить страницу.

Лекция 21 ⏮

Создание презентаций в LaTeX'е

Требования к современным электронным презентациям. Пакеты Beamer, Prosper, PowerDot для создания высококачественных pdf-презентаций в среде LaTeX

В дополнение к докладам или отчетам теперь часто приходится делать электронные презентации, содержащие рисунки, таблицы и формулы. Современные электронные презентации должны отвечать следующим требованиям:

- Мобильность (возможность одинакового воспроизведения на разных компьютерах и платформах).
- Возможность использовать для создания свободно распространяемое программное обеспечение.
- Открытие сразу в полноэкранный режим.
- Строгость, но «стильность» оформления.
- Наличие титульного листа.
- Навигация по разделам и слайдам.
- Наличие на слайдах рабочих ссылок на ресурсы Интернет и на локальные документы разных типов.
- Наличие эффектов перехода и анимации объектов.
- Возможность использования звука и анимации на слайдах.

Высококачественную презентацию, удовлетворяющую первым двум требованиям, обеспечивает формат pdf. Файл презентации с расширением .pdf может быть получен разными средствами, например, путем экспорта презентации из офисного пакета. Но не все эффекты, работающие в офисном пакете, сохраняются при экспорте в pdf.

Средствами LaTeXвозможна генерация файла презентации в формате pdf. При этом презентации, созданные при помощи современных пакетов LaTeX'а, успешно конкурируют с презентациями, созданным с помощью программы PowerPoint.

Для создания электронных презентаций в LaTeX'е существует ряд специальных пакетов, таких как Beamer, Prosper, PowerDot, PdfScreen, PPower4, PdfSlide и другие. При этом почти все приемы оформления текста в LaTeXбудут работать и в презентациях, а значит из статьи или отчета pdf-файл презентации сделать очень просто.

Мы остановимся на классических и современных наиболее зарекомендовавших себя пакетах. Рассматриваемые в рамках данной лекции пакеты входят в полный дистрибутив MikTex 2.7 и создают стандартными средствами этого дистрибутива pdf-файлы, не требующие какой-либо обработки при помощи дополнительного программного обеспечения.

Далее ограничимся рассмотрением основного набора возможных функций, вполне достаточного для подготовки презентаций научного и научно-популярного содержания.

Независимо от конкретного пакета, для получения pdf-файла презентации необходимо:

- Дважды обработать исходный файл имяфайла.tex при помощи pdflatex, т.е. выполнить дважды команду pdflatex имяфайла.

- Если в исходном файле присутствует postscript графика, то имяфайла.tex обрабатывается при помощи latex, полученный dvi-файл командой dvips конвертируется в имяфайла.ps, а уже затем в файл с расширением pdf (для этого используется команда ps2pdf). Цепочка latex-dvips-ps2pdf позволяет корректно вставить рисунки eps-формата и рисунки, выполненные с помощью пакета pstricks.

Наслаждаться эффектами полученной pdf-презентации можно, используя Acrobat Reader или другие подобные программы.

1. Пакет Beamer

Для создания высококачественных электронных презентаций можно использовать достаточно современный класс beamer.

Молодой (чуть более трех лет) пакет beamer разработан Тиллем Тантау (Till Tantau) и очень широко распространен в Америке, где де-факто стал стандартом для создания презентаций. В нем заложено огромное количество возможностей. В beamer возможно задание эффектов «проявления» текста на слайдах, эффектов анимации и перехода слайдов, а также большого числа вариантов оформления с прекрасным дизайном. Кроме того, в beamer при помощи пакета hyperref обеспечиваются ссылки как на объекты слайдов, так и на внешние объекты. Пакет beamer почти идеален для включения в презентацию анимации; музыка, кино и другие мультимедиа файлы легко присоединяются к создаваемому файлу.

Исходная документация по пакету Beamer занимает около 214 листов. После установки полной версии Miktex 2.7 ее можно найти в директориях Miktex2.7/doc/latex/beamer и Miktex2.7/doc/latex/powerdot.

- Пример типичной структуры презентации в beamer

1. Титульный лист.
2. Слайд с оглавлением по разделам.
3. Два раздела.
4. В одном разделе –– слайд со ссылками на слайды раздела, слайд с «проявлением» объектов и слайд с «выезжающей» таблицей, для которой имеются эффекты перехода;
5. В другом разделе — слайд со ссылками на слайды раздела, слайд со ссылками на различные документы (локальный html, рисунок, электронную таблицу, веб-страницу и внешний звуковой файл) и слайд

с формулами, «проявляющимися» по щелчку мыши. Кроме того, для последнего слайда задан «эффект перехода» в виде «открывающихся жалюзей».

Всего в такой презентации получится восемь слайдов. Внешние объекты разного типа можно хранить в отдельных папках (например, с одноименными названиями `html, images, docs, sounds`).

- Преамбула LaTeXдокумента.

В преамбуле, как обычно, объявляется класс документа, подключаются необходимые пакеты с нужными опциями и определяются сведения для титульного листа. Она может быть, например, такой:

```
\documentclass[12pt]{beamer}
\usepackage[T2A]{fontenc}
\usepackage[cp1251]{inputenc}
\usepackage[russian]{babel}
\usepackage{graphicx}
%
\hyperbaseurl{.}
\hypersetup{pdfpagemode=FullScreen,colorlinks=true}
%
\setbeamercovered{transparent}
\usecolortheme{rose}
\usetheme{JuanLesPins}
\useinnertheme{rounded}
\useoutertheme{smoothbars}%
%
\title{Некоторые возможности \texttt{beamer}}
\date{октябрь 2008}
\author{\small О.В. Знаменская}
\institute{Сибирский федеральный университет}
\logo{\includegraphics[width=1cm]{logo}}
```

Первые три строки определяют класс документа и возможности использования русского языка и графики.

Класс `beamer` загружает пакет `hyperref` по умолчанию. Но если в документе планируется использовать этот пакет с установками, отличными от стандартных, то их следует передать в качестве необязательного параметра команды выбора класса. Следующие две строки настраивают пакет `hyperref`. А именно, текущий каталог (т.е. каталог с презентацией) объявляется базовым, так что ссылки могут задаваться относительно него. Кроме того, объявляется открытие презентации в полноэкранном режиме и выделение ссылок разными цветами, в зависимости от типа ссылки.

Следующие пять строк определяют стиль оформления презентации.

Команда `\setbeamercovered{transparent}` обеспечивает «проявление» элементов презентации на слайде, следующие команды определяют цветовую гамму и стиль оформления слайдов. Стили презентации в данном пакете шифруются именем университета, в стиле и цветах которого предполагается оформить презентацию.

Последние строки в преамбуле задают содержание титульного листа(если не ставить дату, то `beamer` автоматически подставит текущую дату). Команда `\logo` печатает на слайдах нужный автору логотип (кафедры, факультета, университета или конференции).

- Выбор схемы презентации.

В пакете `beamer` можно выбрать *схему презентации, цветовую гамму, внешнюю и внутреннюю структуру*. Сочетание этих четырех параметров дает возможность получать различные визуальные эффекты.
Несколько возможных схем:

```
\usetheme{Warsaw}       \usetheme{Szeged}
\usetheme{Antibes}      \usetheme{JuanLesPins}
\usetheme{Montpellier}  \usetheme{Berkeley}
\usetheme{Boadilla}     \usetheme{PaloAlto}
\usetheme{CambridgeUS}
```

Несколько возможных цветовых гамм:

```
\usecolortheme[named=SeaGreen]{structure}
\usecolortheme{albatross}   \usecolortheme{crane}
\usecolortheme{wolverine}   \usecolortheme{sidebartab}
```

Внешняя структура:

```
\useoutnertheme{default}   \useoutnertheme{infolines}
\useoutertheme{shadow}     \useoutertheme{split}
\useoutertheme{smoothbars} \useoutnertheme{sidebar}
```

Внутренняя структура:

```
\useinnertheme{default}     \useinnertheme{circles}
\useinnertheme{rectangles}  \useinnertheme[shadow]{rounded}
\useinnertheme{inmargin}
```

Можно начать с такой комбинации параметров:

```
\usecolortheme{rose}    \usetheme{Warsaw}
\useinnertheme{rounded} \useoutertheme{shadow}
```

Внутренняя структура определяет оформление перечней, списков, блоков, теорем и таблиц содержания. В квадратных скобках могут быть перечислены дополнительные параметры. Команды `\useinnertheme` и `\useoutnertheme` можно не использовать, в этом случае структура будет формироваться по умолчанию.

В документации по `beamer` можно найти полный перечень вариантов оформления презентаций.

- Описание структуры презентации.

Сначала целесообразно задавать общую структуру документа (зарезервировать необходимое число слайдов и необходимое число разделов), а уже потом приступать к их наполнению содержанием и определению эффектов.

Структура презентации из двух разделов по два слайда в каждом, с отдельным слайдом содержания и титульной страницей, будет определяться в файле **имяфайла.tex** следующим образом:

```
\begin{document}
\begin{frame}\titlepage\end{frame}
\begin{frame}
\frametitle{Содержание}
\tableofcontents
\end{frame}
\section{Раздел1}
\begin{frame}
\frametitle{Содержание раздела1}%позже будет
\end{frame}
\begin{frame}[label=p1s1]
\frametitle{Слайд1p1}%будет содержание слайда
\end{frame}
\begin{frame}[label=p1s2]
\frametitle{Слайд2p1}%будет содержание слайда
\end{frame}
\section{Раздел2}
\begin{frame}
\frametitle{Содержание раздела2}%позже будет
\end{frame}
\begin{frame}[label=p2s1]
\frametitle{Слайд1p2}%будет содержание слайда
\end{frame}
\begin{frame}[label=p2s2]
\frametitle{Слайд2p2}%будет содержание слайда
\end{frame}
\end{document}
```

Раздел документа `\section{название раздела}` продолжается до тех пор, пока не объявлен следующий раздел или не кончится документ.

Напомним, что если раздел объявлен с использование символа $*$, то ему не будет присвоен номер и он не попадет в оглавление.

Слайд занимает отдельную страницу презентации и оформляются при помощи окружения `\begin{frame}[параметры, например, label=метка кадра]Заголовок...\end{frame}`. Может иметь заголовок и метку. Метка нужна для возврата на слайд по командам условного перехода. Слайд может содержать несколько блоков (оверлэев). Заголовки слайдов объявляются командой `\frametitle{текст}` и отображаются в соответствии с темой слайдов. Метки `[label= ... ]` слайдов необходимы для гиперссылок на них или их блоки (оверлеи).

Команда `\titlepage` в первом слайде создает титульный лист презентации на основе данных из преамбулы.

Команда `\tableofcontents` на втором слайде формирует нумерованный список разделов, причем автоматически создаются ссылки на начальные слайды этих разделов. Кроме того, `beamer` добавляет собственные средства навигации внизу в правой части слайда.

Все остальные слайды пока пустые, на них нет ничего кроме названия.

При заполнении слайдов недо учитывать, что блок автоматически выделяется в зависимости от выбранной схемы презентации. Все стандартные `tex`-окружения оформляются в виде блоков по умолчанию. В виде выделенного блока может быть красиво оформлена любая формула.

Нужно учесть, что пакет не осуществляет автоматической разбивки текста по кадрам. Если текст не войдет в отображаемое пространство, он будет не виден. Текст в кадре выравнивается по левому краю, системы переносов нет. Командами `\begin{center}` или `\end{center}` кадр или блок можно центрировать. По мере заполнения каждый слайд неплохо дополнять комментариями, чтобы не запутаться и не забыть, что отображается на слайдах.

- Слайд со ссылками на слайды.

Потребуем, чтобы в презентации первый слайд в каждом разделе содержал только список слайдов данного раздела. Пусть этот список будет

маркированным (без номеров). Для формирования маркированного списка можно использовать окружение `itemize`. Для формирования ссылок на каждый слайд можно использовать команду `hyperlink` и метки, присвоенные слайдам ранее.

Первым аргументом команды `\hyperlink` является метка целевого слайда, вторым –– произвольный текст (не обязательно названия слайдов).

Пример команд для для слайда презентации, удовлетворяющей этим требованиям (у нас первого слайда второго раздела), приведен ниже.

```
\section{Раздел 2}
\begin{frame}
\frametitle{Содержание раздела 2}
\begin{itemize}
\item \hyperlink{p2s1}{Слайд\,2.1: гиперссылки}
\item \hyperlink{p2s2}{Слайд\,2.2: формулы и
эффекты перехода}
\end{itemize}
\end{frame}
```

- Использование `overlay` и `multimedia` эффектов внутри текста.

Конструкция, приведенная ниже, позволяет отображать список, каждый последующий элемент которого отображается на месте предыдущего (в одном слайде).

```
\begin{frame}
Основной текст
\begin{itemize}
\only<1>{\item Первый элемент списка}
\only<2>{\item Второй элемент списка}
\only<3>{\item Третий элемент списка}
\end{itemize}
\end{frame}
```

Следующая конструкция позволяет последовательно отображать элементы списка при показе.

```
\begin{enumerate}[<+->]
\item{ Первый элемент списка }
\item{ Второй элемент списка }
\item{ Третий элемент списка }
\item{ Четвертый элемент списка}
\end{enumerate}
```

При подключении пакета `multimedia` при помощи команды `\usepackage{multimedia}` в презентации можно проигрывать звуковые и видео-файлы.

- Слайд с «проявлением» текста.

Чтобы создать слайд с текстом, проявляющимся по щелчку мыши, нужно в преамбуле установить режим `transparent` и использовать команду `\pause` как показано ниже.

Вообще говоря, команда `\pause` позволяет задать интервал времени перед проявлением объекта. Однако в научных и деловых презентациях вообще не рекомендуется использовать автоматические переходы.

```
\begin{frame}[label=p1s1]
\frametitle{Проявление текста}
\pause
Этот текст <<проявляется>> после
первого щелчка мышью \\
\pause
Этот после второго
\end{frame}
```

- Слайд с формулами и эффектами перехода.

Формулы набираются как обычно в LaTeXс использованием соответствующих пакетов. Ниже рассматривается пример, когда для слайда установлен эффект перехода `\transblindshorizontal` (горизонтально открывающиеся «жалюзи»), который срабатывает как для первого блока, так и для второго.

Листинг команд слайда:

```
\begin{frame}[label=p2s2]
\frametitle{Формулы и эффекты перехода}
\transblindshorizontal
%
\begin{block}{Логарифмическое отображение  Гаусса}
{\small \begin{equation*}
    \gamma: \, V \rightarrow \mathbb{CP}_{n-1} , \ \
    (z_1, \dots, z_n) \longmapsto (z_1 P^{\prime}{z_1}:
    \dots : z_n {P^{\prime}}{z_n}),
  \end{equation*}
    где $P^{\prime}{z_j} = \partial P / \partial z_j$.}
\end{block}
\pause
%
\begin{block}{Моментное отображение
$\mu: \mathbb{CP}_{n}\rightarrow \Sigma_n$}
{\small \begin{equation*}
    (Z_0:\dots:Z_n)\longrightarrow
    \frac{(|Z_0|,\dots,|Z_n|)}{|Z_0|+\dots+|Z_n|},
     \end{equation*}
 где $\Sigma_n=\bigg \{t\in\mathbb R^{n+1}:
 t_j\geq 0, \sum\limits_{j=0}^n t_j=1\bigg \}$
 --- стандартный симплекс.}
\end{block}
\end{frame}
```

На рисунке приведен вид презентации после перехода на этот слайд и после щелчка мыши на нем.

Заметим, что красивые переходы нужны для того, чтобы переключить внимание слушателей. Обычно они используются для подчеркивания мысли, для пробуждения уставших слушателей (а иногда для создания внешних эффектов, отвлекающих от содержания презентации). Применять спе-

цэффекты на докладах в среде специалистов рекомендуется в ограниченных дозах.

- Слайд с «выползающей» таблицей.

Эффекты анимации в Beamer означают смену элементов слайда (оверлеев). Если установлен режим ручного перехода от слайда к слайду, то анимация будет состоять в изменении позиции или размера оверлея по щелчку мыши.

Рассмотрим пример изготовления слайда со структурным блоком, содержащим таблицу, число строк в которой увеличивается от двух до четырех. Листинг команд слайда:

```
\begin{frame}[label=p1s2]
\frametitle{<<Выползающая>> таблица}
\only<1>{
\begin{beamerboxesrounded}[shadow=true,width=7.5cm]
{Блок с таблицей}
\begin{tabular}{|r|c|l|} \hline
\textbf{Столбец1}&\textbf{Столбец2}&\textbf{Столбец3}\\
\hline Данные11 & Данные21 & Данные31 \\ \hline
\end{tabular}
\end{beamerboxesrounded}
}
\only<2>{
\begin{beamerboxesrounded}[shadow=true,width=7.5cm]
{Блок с таблицей}
\begin{tabular}{|r|c|l|} \hline
\textbf{Столбец1}&\textbf{Столбец2}&\textbf{Столбец3}\\
\hline Данные11 & Данные21 & Данные31 \\
\hline Данные12 & Данные22 & Данные32 \\ \hline
\end{tabular}
\end{beamerboxesrounded}
}
```

```
\only<3>{
\begin{beamerboxesrounded}[shadow=true,width=7.5cm]
{Блок с таблицей}
\begin{tabular}{|r|c|l|} \hline
\textbf{Столбец1}&\textbf{Столбец2}&\textbf{Столбец3}\\
\hline Данные11 & Данные21 & Данные31 \\
\hline Данные12 & Данные22 & Данные32 \\
\hline Данные13 & Данные23 & Данные33 \\
\hline
\end{tabular}
\end{beamerboxesrounded}
\beamerbutton{Кнопочка!
(Возвращает к содержанию Раздела 1)}
}\end{frame}
```

Для создания самой таблицы используется окружение `\tabular`.

Слайд, заданный командами выше, состоит из трех оверлеев с условием, что каждый из них полностью заменяется следующим по щелчку мышью. Это определяется с помощью окружения `\only<N>{...}`, где N — номер оверлея. По щелчку мыши оверлей сменяется следующим. Если какой-то объект должен быть виден все время, то в таком случае надо писать, например, `\begin{block}<1-3>{Какой-то блок}...\end{block}`. Такая декларация обеспечит наличие блока на протяжении показа оверлеев с первого по третий. При переходе от оверлея к оверлею номер слайда не изменяется.

Для создания структурного блока здесь использована структура `beamerboxesrounded` с параметрами, определяющими наличие тени и ширину блока. Здесь выбран `beamerboxesrounded`, поскольку для него можно указывать ширину явно. В случае использования конструкции `\begin{block}...\end{block}` блок получится во всю ширину слайда и справа от таблицы будет много пустого места.

Заметим, что реальный размер слайдов составляет 128×98 мм, поэтому все явно указываемые размеры должны быть соотнесены с этим масштабом.

Команда `\beamerbutton{произвольный текст}` создает «кнопочку» с соответствующей надписью.

- Слайд с гиперссылками.

Окружение `enumerate` создает нумерованный список, а команда `\href` служит для создания гиперссылки. Опция `run:` обеспечивает открытие файла приложением, по умолчанию связанным с данным типом файла, вне зависимости от имеющегося в системе программного обеспечения.

Различные типы ссылок по умолчанию отображаются разными цветами, независимо от выбранной цветовой схемы, а именно, ссылки на слайды презентации -- красным, на объекты с использованием метода `run:` — голубым, а на ресурсы Интернет -- малиновым.

Листинг команд слайда:

```
\begin{frame}[label=p2s1]
\frametitle{Гиперссылки}
\begin{enumerate}
\item \href{run:html/sample.html}%
{Локальный HTMLдокумент}
\item \href{run:images/sample.png}%
{Картинка}
\item \href{run:docs/sample.ods}%
{Электронная таблица в OpenOffice}
\item \href{http://ice.spb.ru}%
{Ссылка на сайт}
\item \href{run:sounds/sample.ogg}%
{Аудиоролик}
\end{enumerate}
\end{frame}
```

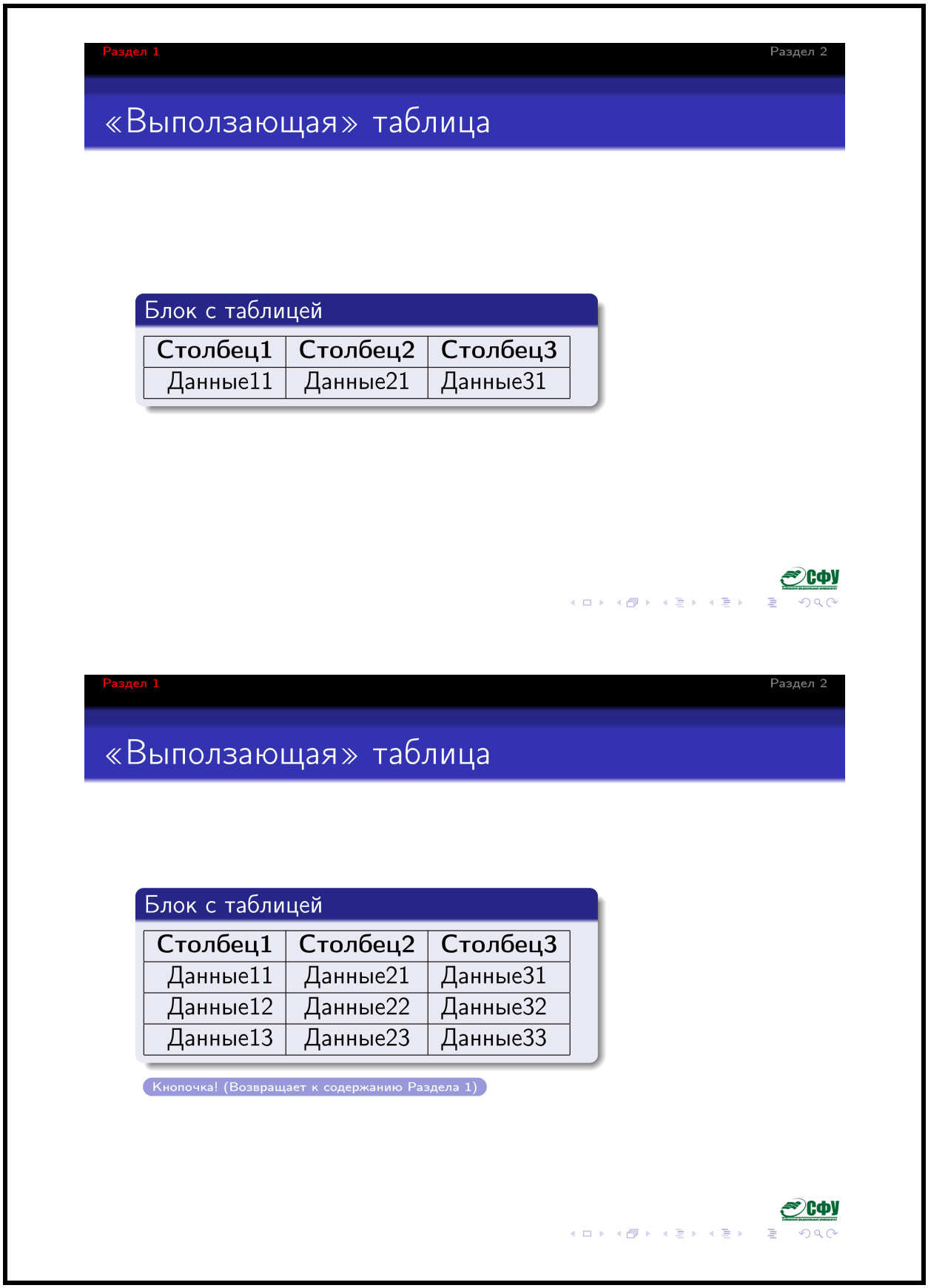


Рис: 21.1. Некоторые результаты применения описанных выше команд (слайд с формулами и эффектами перехода).

Раздел 1 Раздел 2

Формулы и эффекты перехода

Логарифмическое отображение Гаусса

$$\gamma : V \to \mathbb{CP}_{n-1}, \quad (z_1, \dots, z_n) \longmapsto (z_1 P' z_1 : \dots : z_n P' z_n),$$

где $P' z_j = \partial P / \partial z_j$.

Моментное отображение $\mu : \mathbb{CP}_n \to \Sigma_n$

$$(Z_0 : \dots : Z_n) \longrightarrow \frac{(|Z_0|, \dots, |Z_n|)}{|Z_0| + \dots + |Z_n|},$$

где $\Sigma_n = \left\{ t \in \mathbb{R}^{n+1} : t_j \geq 0, \sum_{j=0}^{n} t_j = 1 \right\}$ — стандартный симплекс.

Раздел 1 Раздел 2

Формулы и эффекты перехода

Логарифмическое отображение Гаусса

$$\gamma : V \to \mathbb{CP}_{n-1}, \quad (z_1, \dots, z_n) \longmapsto (z_1 P' z_1 : \dots : z_n P' z_n),$$

где $P' z_j = \partial P / \partial z_j$.

Моментное отображение $\mu : \mathbb{CP}_n \to \Sigma_n$

$$(Z_0 : \dots : Z_n) \longrightarrow \frac{(|Z_0|, \dots, |Z_n|)}{|Z_0| + \dots + |Z_n|},$$

где $\Sigma_n = \left\{ t \in \mathbb{R}^{n+1} : t_j \geq 0, \sum_{j=0}^{n} t_j = 1 \right\}$ — стандартный симплекс.

Рис: 21.2. Некоторые результаты применения описанных выше команд (слайд с «выползающей» таблицей).

- Выбор цвет при создании презентации.

Этот вопрос рассмотрим на конкретных примерах.

Команда `\beamertemplateshadingbackground{red!5}{yellow!15}` задает цветовой тон слайда.

Цветной заголовок блока можно определить с помощью команды `\begin{block}{\color{orange!50}{Заголовок}} \end{block}`.

Выделение слова в тексте цветом, принятым по умолчанию для выделения для данной схемы презентациио существляется при помощи команды `\alert{Текст}`.

Выделение слова в тексте нужным цветом можно сделать с помощью `\textcolor{violet}{Текст}`.

В некоторых командах можно указывать интенсивность выбранного цвета или смешивать цвета.

- Вставка рисунков в презентацию.

Каждый рисунок лучше вставлять в отдельный кадр следующим образом:

```
\begin{frame}
\begin{pspicture}(0,0)(9,3)
Команды формирования рисунка
\end{pspicture}
\end{frame}
\begin{frame}
\begin{picture}(150,110)
Команды формирования рисунка
\end{pspicture}
\end{frame}
\begin{frame}
\begin{center}
\includegraphics[height=5cm]{1.eps}
\end{center}
\end{frame}
```

При использовании пакета PGF возможности по работе с графикой расширяются:

```
\begin{frame}[plain]
\frametitle{\small Графика}
\begin{colormixin}{80!white}
\begin{pgfpictureboxed}{0cm}{0cm}{10cm}{7cm}
\pgfputat{\pgfxy(0,0)}{\pgfbox[left,base]%
{\pgfimage[width=2cm]{newimm}}}
\pgfputat{\pgfxy(0,7)}%
{\pgfbox[left,top]%
{\pgfimage[width=2cm]{newimm}}}
\pgfputat{\pgfxy(10,0)}%
{\pgfbox[right,base]%
{\pgfimage[width=2cm]{newimm}}}
\pgfputat{\pgfxy(10,7)}%
{\pgfbox[right,top]%
{\pgfimage[width=2cm]{newimm}}}
\pgfputlabelrotated{.33}{\pgfxy(4,2)}%
{\pgfxy(8,5)}{5pt}%
{\pgfbox[center,center]
{\textcolor{red}{Название рисунка}}}
\pgfputlabelrotated{.33}{\pgfxy(4,4)}%
{\pgfxy(6,1)}{5pt}{\pgfbox[center,center]
{\textcolor{blue}{Легко работаем с графикой!}}}
\end{pgfpictureboxed}
\end{colormixin}
\end{frame}
```

Пакет beamer имеет огромное количество настроек и параметров, которые вы можете задать, если захотите. Можно менять темы оформления, цвета и размеры шрифтов, подключать другие гарнитуры, изменять стиль отображения заголовков. Всё это позволяет точно подстраивать результат под разнообразные нужды.

2. Пакет PowerDot

Данный пакет на настоящий момент кажется наиболее оптимальным по отношению времени изучения к набору заложенных в нем возможностей.

Рассмотрим основные команды преамбулы файла класса PowerDot на примере конкретного файла презентации.

```
\documentclass[size=12pt,style=fyma,mode=present,%
paper=a4paper,clock]{powerdot}
\usepackage[english]{babel}
\usepackage{amsfonts,amssymb,amsmath,mathrsfs}
\usepackage{listings}
\usepackage{pstricks,pst-node,pst-text,pst-3d,pst-grad}
\usepackage{moreverb,epsfig,color,subfigure}
\definecolor{brred}{rgb}{1.,.1,.1}
\title{PowerDot example}
\author{Author}
```

Необязательные аргументы команды \documentclass:

- size=12pt — размер шрифта (для презентации довольно маленький);
- style=fyma — определяет форму и цвета́ презентации. Вместо fyma можно использовать стили default, simple, tycja, ikeda, ciment, elcolors, aggie, husky, sailor, upen, bframe horatio, paintings, klope, jefka, pazik;
- mode=present — опция используется для вывода презентации на экран. Для печати и заметок используются опции mode=print и mode=handout;
- paper=a4paper — задает размер изображения, возможные варианты paper=screen и paper=letterpaper;

- clock — появление электронных часов в правом верхнем углу.

Другие опции можно найти в документации к пакету.

Далее в преамбуле `tex`-файла, как обычно, подключаются вспомогательные пакеты. В случае простого оформления слайдов презентации (без листингов, таблиц, сложных графических объектов и формул) можно ограничиться только подключением пакета `babel` с указанием языка, на котором написан слайд.

Команда `\definecolor{brred}{rgb}{1.,.1,.1}` определяет собственный цвет.

`\title` (название презентации) и `\author` (автор презентации) как правило пишут до команды `\begin{document}` но можно указывать их и в теле документа перед командой `\maketitle`.

Далее начинается «тело» документа. Отдельные слайды презентации помещаются в окружение `\begin{slide}` ... `\end{slide}`:

```
\begin{document}
\maketitle
\begin{slide}{First slide}
Hello \LaTeX!!!
\end{slide}
\begin{slide}[trans=Glitter/Di0]{Second slide}
It's my second slide.
\end{slide}
```

По умолчанию, все заголовки слайдов помещаются слева на экране для быстрого перехода между ними. Если необходимо использовать всю ширину экрана, то вывод списка слайдов можно отменить, используя заголовок типа `\begin{slide}[toc= ,bm=  ]{First slide}`

Переходы между слайдами задаются опцией `trans`, которая принимает значения `Split, Blinds, Box, Wipe, Dissolve, Glitter, Replace, Fly, Push, Cover, Uncover, Fade`. Переходы будут видны только при полноэкранном режиме работы Adobe Reader.

Вот немного более сложный слайд:

```
\begin{slide}[trans=Wipe]{The quest for $\pi$}
\begin{itemize}
\item The following formula computes $8$ correct
digits per iteration (Ramanujan):
\pause
\item
\begin{small}
\begin{equation*}
{\color{red} \frac{1}{\pi}}=\sum_{n=0}^\infty
\frac{(\frac{1}{4})_n(\frac{2}{4})_n
(\frac{3}{4})_n}{n!^3}
\bigl(2\sqrt{2}(1103+26390n)\bigr)
\frac{1}{(99^2)^{2n+1}}
\end{equation*}
\end{small}
\end{itemize}
\end{slide}%
```

Для различных цветов используется команда `\color{имяцвета}`, а для переходов «вручную» — команда `\pause`.

Следующий слайд — более сложный пример работы с цветом.

Кроме команды `\pause` слайд разбить на части и показывать эти части в необходимом порядке. Для этого служит команда `\onslide{m,n}`. Цифры `m` и `n` в скобках указывают на сколько частей разбит слайд и когда видна та или иная часть слайда.

```
\begin{slide}{Degree Of Anonymity}
\begin{itemize}
\item<1-> Defines a degree of anonymity $d$ based on
entropy of the system
\item<2-> The entropy of the system is
\[\onslide*{3}{\textcolor{brred}{H(X)}}
```

```
\onslide*{-2,4-}{H(X)}=-\sum_{i=1}^{
\onslide*{4}{\textcolor{brred}{N}}
\onslide*{-3,5-}{N}}
\left(
\onslide*{5}{\textcolor{brred}{p_i}}
\onslide*{-4,6-}{p_i}
\lg(\onslide*{5}{\textcolor{brred}{p_i}}
\onslide*{-4,6-}{p_i})\right)\]
\begin{itemize}
\item \onslide+{3}{The entropy for the system}
\item \onslide+{4}{The number of nodes}
\item \onslide+{5}{The probability associated
with node $i$}
\end{itemize}
\item<6-> And the maximal is $HM = \lg(N)$
\item<7-> Finally $d = \frac{H(X)}{HM}$
\end{itemize}
\end{slide}
```

Как и обычный `tex`-файл презентацию можно структурировать, выделяя разделы и подразделы.

```
\section{First section}
\begin{slide}{Dynamical system}
\onslide{1,2}{
Dynamical systems on $\mathscr M$ having enough
functionally independent integrals of motion
$H_1,\ldots H_{n}$ in involution with respect to
both Poisson brackets \[\{H_i,H_j\}_0=
\{H_i,H_j\}_1=0\,.\] will be called {\color{red}
bi-integrable systems}.\vskip0.5cm}
```

```
\onslide{2}{
\color{black}\textbf{Example:}
If the vector fields $X_{H_i}$ are bi-hamiltonian
vector fields, which form a so-called anchored
Lenard-Magri sequence \[ P_0dH_1=0,\qquad
X_{H_i}=P_0dH_i=P_1dH_{i-1}, \qquad P_1dH_n=0.\]
then integrals of motion $H_1,\ldots H_{n}$ are
in the bi-involution.}
\end{slide}
```

Разместим на слайде графические объекты и текстовые команды:

```
\begin{slide}[method=file]{Taylor}
\begin{lstlisting}[escapechar=|,mathescape]
e^x |\pause| = 1 + |\pause|x +
|\pause|x^2/2! + |\color{red}{\ldots}|
|\pause| e^x |\pause| = 1 + |\pause|x +
|\pause|$\rnode{A}{\frac{x^2}{2!}}$ +
|\color{red}{\ldots}|
\end{lstlisting}
\pause
\vspace{.1in} Notice, \rnode{B}{this}
fraction is in \verb|mathmode|.
\onslide*{11-}{
\ncarc[linecolor=blue,arcangle=-5]{->}{B}{A}}\pause
\vspace{.1in}
\end{slide}
```

Можно включить в презентацию и более сложные конструкции, например, графические примитивы пакета `pstrics`. Об этом можно прочитать в документации к пакету на CTAN.

В заключение остается дать команду `\enddocument`, и презентация будет готова.

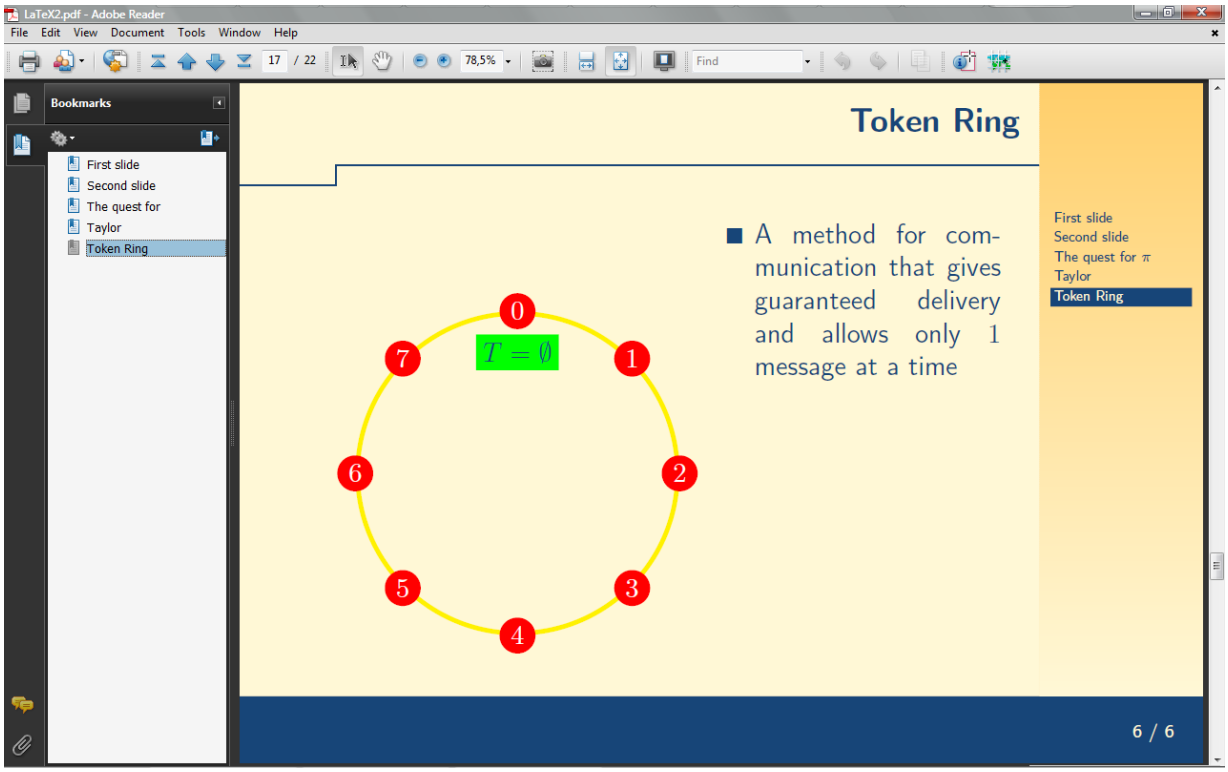


Рис: 21.3. Образец слайда презентации в PowerDot

Для эксперимента можно менять стили презентации в преамбуле и другие опции.

3. Пакет Prosper

Пакет Prosper довольно широко распространен в Европе.

Компиляция pdf-файла происходит так же, как и для остальных пакетов.

Тем презентаций в пакете не так много, как в beamer, но их тоже можно менять. Их названия: azure, slideBW, draft,contemporain, nuancegris, troispoints, lignesbleues, darkblue, alienglow, autumn, light.

Ниже приводится пример tex-файла, в котором комментарии, как обычно, помечены знаком %.

```
\documentclass[pdf,colorBG,slideColor,azure]{Prosper}
%Другие опции в преамбуле тоже есть!
\usepackage{amsmath}
\logo{\includegraphics[width=1cm]{logo}}
\begin{document}
```

Пусть в этой презентации слайд содержит два перехода (наложения):

```
\overlays{2}{\begin{slide}[Split]{The quest for $\pi$}
\begin{itemstep}
   \item The following formula computes $8$ correct
digits per iteration (Ramanujan):
   \item\begin{small}   \begin{equation*}
\frac{1}{\pi}=\sum_{n=0}^\infty \frac{(\frac{1}{4})_n
(\frac{2}{4})_n(\frac{3}{4})_n}{n!^3} (2\sqrt{2}
(1103+26390n))\frac{1}{(99^2)^{2n+1}}  \end{equation*}
\end{small}\end{itemstep}\end{slide}}  \ end{document}
```

В команде `\begin{slide}[стиль]{текст}` стиль переходов определяется опциями: `Split, Blinds, Box, Wipe, Replace, Dissolve.`

А вот так можно создать более сложный слайд:

```
\overlays{7}{
\begin{slide}[Replace]{The color theme}
 \begin{itemstep}
 \item A first item
 \item A second item
     \begin{itemstep}
     \item A sub item \begin{itemstep}
         \item \color{blue} A sub sub item
         \item \color{red} Another sub sub item
                     \end{itemstep}
     \item Back to sub item
     \end{itemstep}
 \item Last item
 \end{itemstep}
\end{slide} }
```

Подробности описаны в соответствующей документации к пакету, входящей в дистрибутив MikTeX 2.7.

4. Несколько полезных советов

Качественная презентация — это не только произведение искусства, но и мощное средство обеспечения взаимопонимания между докладчиком и слушателями, поэтому не следует жалеть о потраченных на наведение блеска минутах. Готовя презентацию, не следует забывать об эмпирически выведенных правилах, предложенных Е. Балдиным `http://wiki.linuxformat.ru/`:

- Один слайд требует не меньше одной минуты.
- Один слайд «со смыслом» требует не менее пяти минут.
- Времени всегда не хватает.
- Не следует помещать в презентацию больше слайдов, чем позволяет время. Перебор по времени только раздражает слушателей.
- Каждый слайд должен иметь свой заголовок (`\frametitle`).
- В один слайд можно поместить около 20-40 слов, но заведомо не больше 80.
- Полезно использовать `block, theorem, proof и example`. Эти окружения структурируют текст и помогают выделять основные мысли.
- Для разных аудиторий правила могут отличаться.

Лекция 22 ⏮

Электронные тесты и генерация вариантов заданий

Средства acrotex для создания электронных тестов. Пакет fp для генерации вариантов учебных заданий

Все, кто в своей деятельности сталкивается с необходимостью создания набора тестовых и контрольных заданий, хорошо понимают, насколько важны различные средства, облегчающие эту работу. В настоящей лекции мы рассмотрим некоторые возможности LaTeX в создании интерактивных тестов и генерации учебных заданий со случайными входными данными.

1. Создание электронных тестовых заданий средствами acrotex

Набор пакетов `acrotex` для изготовления в LaTeX электронных документов в формате `PDF`, в том числе разного рода учебных и тестовых заданий, был создан Д.П. Стори (D. P. Story) в университете Акрон (Akron) в 1999-2000 гг. и обновлен летом 2007 г. (вся документация и самый новый дистрибутив находятся в свободном доступе на персональной странице `acrotex`: http://www.math.uakron.edu/~dpstory/webeq.html.

1.1. Пакеты и их подключение

Для того, чтобы сгенерировать электронные тесты разного типа средствами макропакета `acrotex`, предпоследняя версия которого входит в современный дистрибутив MikTeX2.7, нужно в преамбуле документа подключить два обязательных пакета: пакет `web` и пакет `exerquiz`, написав команды:

```
\usepackage[<опции>]{web}
\usepackage[<опции>]{exerquiz}
```

Целью пакета `Web` является создание оформления страницы для документов, предназначенных для презентации на экране (неважно, для лекции, конференции или публикации в Internet). Такие документы (обычно)

не предназначаются для распечатки; соответственно, оформление страницы в некотором смысле оптимизировано для просмотра на экране.

Пакет `Web` переопределяет команды `\maketitle` и `\tableofcontents` в более дружественные: он раскрашивает заголовки параграфов и вставляет `\bullets` (•) на уровне `\subsubsection`, что выглядит на экране более привлекательно. В пакет также включены дополнительно определенные устройства для навигации — полочка и иконки с направлениями.

Имеются (необязательные) опции для подключения наиболее употребимых драйверов: `dvipsone, dvips, dvipdfm` и `pdftex`. В случае обработки исходного файла при помощи `pdflatex`, указывается опция `pdftex`.

Опция `language` переопределяет некоторые зависящие от языка элементы пакета на другие языки. Ее допустимые значения: `dutch, french, german, italian, norsk, russian spanish, dansk, polish, finnish, czech, catalan, and brazil`.

Опция переформатирования для распечатки на бумаге: `forpaper`. Используя `forpaper`, можно сделать бумажную копию упражнений (см. справочное руководство `Webeqman.pdf`).

Пакет `Web` задуман для демонстрации на экране учебных лекций или научно-технических материалов на конференциях. Для этих целей достаточно декларации стандартного класса `article`.

Пример типичной шапки документа:

```
\documentclass{article}
\usepackage{amsmath}\usepackage{literat}
\usepackage[cp866]{inputenc}
\usepackage[russian]{babel}
\usepackage[pdftex,russian]{web}
\usepackage[pdftex]{exerquiz}
\title{Название документа}
\author{Автор}
```

```
\subject{Назначание}
\keywords{Ключевые слова}
\university{Название университета}
\email{email}
\version{версия}
\copyrightyears{дата}
```

Пакет exerquiz предназначен для изготовления интерактивных файлов в формате PDF, содержащих разного рода задания, тесты и упражнения. Он может быть также использован для изготовления электронных форм. Далее рассмотрим подробнее примеры применения окружений из этого пакета.

Окружение exercise пакета exerquiz облегчает изготовление электронных упражнений. Рассмотрим несколько шаблонов окружений, определенных в exerquiz.

Наиболее интересна *-разновидность окружения exercise, Ее использование Позволяет включить в упражнение нескольких частичных вопросов. Следующие примеры упражнений иллюстрируют применение этой опции:

```
\begin{exercise}*\label{ex:parts}
Предположим, что частица движется вдоль оси $s$ и
местоположение частицы движущейся вдоль оси $s$
в любое время  $t$ вычисляется по формуле $s=t^2-5t+1$.
\begin{parts}
\item[h]\label{item:part} Вычислите скорость
        $v$ частицы при любом значении $t$.
\begin{solution}
Здесь приводится решение первого задания
\end{solution}
\item Вычислите ускорение $a$ частицы при любом
      значении $t$.
```

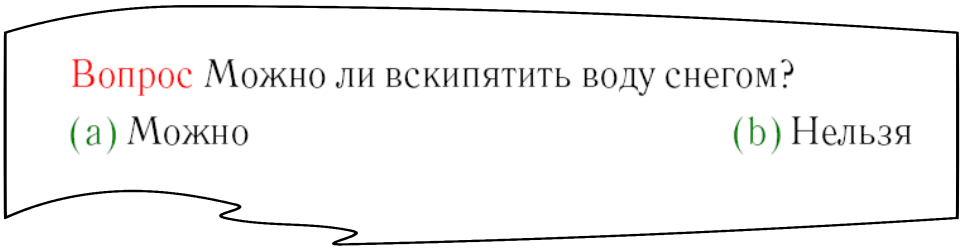
```
\begin{solution}
Здесь приводится решение второго задания
\end{solution}
\end{parts}
\end{exercise}
```

Ссылки могут быть сделаны к определённым частям упражнения; например,

```
\hyperref[item:part]{Exercise~\ref*{ex:parts}%
(\ref*{item:part})}.
```

Окружение `shortquiz` пакета `acrotex` позволяет конструировать тестовые задания с *выбором вариантов ответов*. Приведем пример такой организации заданий, что нажатие мышью на правильный вариант отсылает к пояснению ответа, а нажатие на неправильный вариант не вызывает никакого действия:

```
\begin{shortquiz}
Можно ли вскипятить воду снегом?
\begin{answers}[qz:water1]{2}
\Ans1 Можно & \Ans0 Нельзя
\end{answers}
\begin{solution}
Можно <...>
\end{solution}
\end{shortquiz}
```



После просмотра ответа имеется возможность вернуться к формулировке вопроса, нажав «К формулировке вопроса»:

Ответы к тестам

Ответ:

Можно,
Если заполнить флакон водой до половины и погрузить в кипящую солёную воду. Когда вода закипит, надо будет вынуть флакон и закупорить пробкой. Затем переверните флакон, и ждите, пока вода не перестанет кипеть. Выждав этот момент, положите на его донышко немного снегу или даже облейте его холодной водой, вы увидите, что вода закипит.

К формулировке вопроса

Можно объединить несколько таких заданий с выбором ответа в рамках одного вопроса, сохранив принципы работы с ними:

Вопрос Ответьте на следуюшие вопросы на 100%.

1. Можно ли вскипятить воду снегом?
   (a) Можно (b) Нельзя
1. В каком году Колумб переплыл на корабле голубой океан?
   (a) 1490 (b) 1491 (c) 1492 (d) 1493

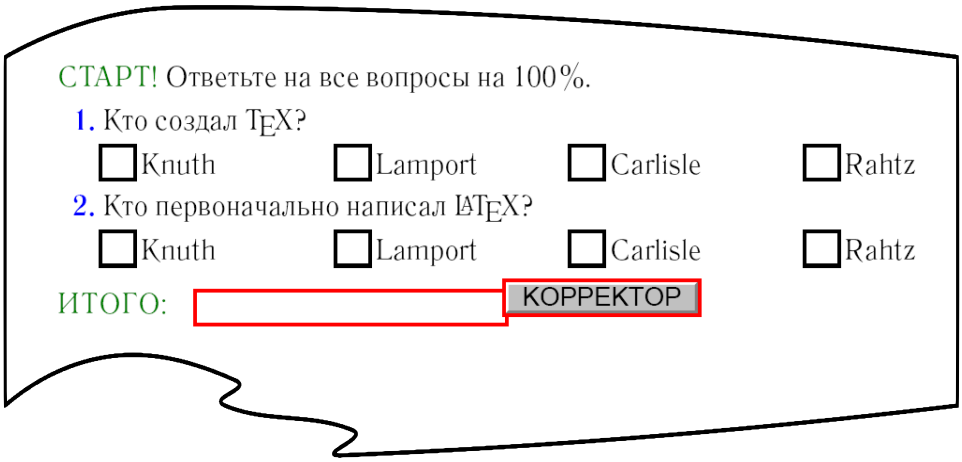
Исходный текст для этого случая использует окружение `shortquiz` и выглядит следующим образом:

```
\begin{shortquiz}
Ответьте на следуюшие вопросы на 100\%.
\begin{questions}
\item Можно ли вскипятить воду снегом?
\begin{answers}[qz:water2]{4}
\Ans1 Можно & \Ans0 Нельзя
\end{answers}
\begin{solution}решение про воду\end{solution}
\end{questions}
```

```
\begin{questions}
\item В каком году Колумб переплыл на
корабле голубой океан?
\begin{answers}[qz:columbus1]{4}
\Ans0 1490 &\Ans0 1491 &\Ans1 1492 &\Ans0 1493
\end{answers}
\begin{solution}
решение про колумба\end{solution}
\end{questions}
\end{shortquiz}
```

Использование окружения `quiz`, обращающегося к *Javascript*, позволяет создавать разного рода электронные тесты как *открытого типа* (требующие вписать правильный ответ и показывающие результат), так и тесты с *выбором вариантов ответа*, позволяющие проверить себя и *после этого узнать правильный ответ*.

Например, следующее задание проверяет знания о людях, которые создали и модернизировали систему TEX.



Заполнение теста начинается после нажатия кнопки «Старт!». При нажатии мышью на соответствующие квадратики в них появляются галочки. Правильные выборы указываются после нажатия кнопки «Итого:». Кнопка «Корректор» вводится для того, чтобы проверить себя (в нижнем окне появляется правильный ответ).

Листинг для этого примера таков:

```
\begin{quiz}*{qz:TeX-f}Ответьте на все вопросы на 100\%.
\begin{questions}
\item Кто создал \TeX?
\begin{answers}4
\Ans1 Knuth &\Ans0 Lamport &\Ans0 Carlisle &\Ans0 Rahtz
\end{answers}
\item Кто первоначально написал \LaTeX?
\begin{answers}4
\Ans0 Knuth &\Ans1 Lamport &\Ans0 Carlisle &\Ans0 Rahtz
\end{answers}
\end{questions}
\end{quiz}\quad
\raisebox{-2pt}{\ScoreField{qz:TeX-f}
\raisebox{3.5pt}{\eqButton{qz:TeX-f}}}
```

При этом дизайн, расположение и название управляющих кнопок могут изменяться пользователем. В заключение приведем пример «работы» тестового задания немного другого типа, но также сделанного с помощью окружения `quiz`:

СТАРТ!

1.3. ОТКРЫТЫ В СТАНДАРТНОЙ ТОПОЛОГИИ НА $\mathbb{R}^2$ ...

а) $x^2 + y^2 \geqslant 4\}$ ☐ да ☐ нет

б) $x^2 + (y-1)^2 \neq 0$ ☐ да ☐ нет

в) $x^2 - y = 0$ ☐ да ☐ нет

г) $x \leqslant 2$ ☐ да ☐ нет

д) $x^2 + y^2 < 1$ ☐ да ☐ нет

ИТОГО: ☐ ОТВЕТ

СТАРТ!

1.3. ОТКРЫТЫ В СТАНДАРТНОЙ ТОПОЛОГИИ НА $\mathbb{R}^2$ ...

а) $x^2 + y^2 \geqslant 4\}$	☑ да	☐ нет
б) $x^2 + (y-1)^2 \neq 0$	☐ да	☑ нет
в) $x^2 - y = 0$	☐ да	☑ нет
г) $x \leqslant 2$	☐ да	☑ нет
д) $x^2 + y^2 < 1$	☑ да	☐ нет

ИТОГО: 3 ОТ 5 ОТВЕТ

СТАРТ!

1.3. ОТКРЫТЫ В СТАНДАРТНОЙ ТОПОЛОГИИ НА $\mathbb{R}^2$ ...

а) $x^2 + y^2 \geqslant 4\}$	☒ да	● нет
б) $x^2 + (y-1)^2 \neq 0$	● да	☒ нет
в) $x^2 - y = 0$	☐ да	✓ нет
г) $x \leqslant 2$	☐ да	✓ нет
д) $x^2 + y^2 < 1$	✓ да	☐ нет

ИТОГО: 3 ОТ 5 ОТВЕТ

2. Возможности пакета fp для генерации учебных заданий

Пакет `fp` (аббревиатура от англ. `floating point`) хорошо подходит для автоматизации создания набора учебных заданий со случайными входными данными. Он разработан М. Мехличем (Michael Mehlich). Данный пакет входит в свободно-распространяемые дистрибутивы MikTEX и TeTEX издательской системы TEX (подробную документацию см. http://www.ctan.org/tex/latex/fp).

2.1. Базовые функции пакета fp

Чтобы использовать пакет `fp` в документе LaTeX 2_ε, необходимо в преамбуле документа вызвать следующую команду:

```
\usepackage[<options>]{fp}.
```

В качестве необязательного параметра <options> можно использовать следующие:

- `nomessages` — в этом случае в `.log` файл и на экран при обработке документа не будут печататься сообщения о вызываемых функциях,
- `debug` — при указании данной опции будут выводиться дополнительные отладочные сообщения.

Команды пакета указываются в теле документа между `\begin{document}` и `\end{document}`. Запретить или разрешить вывод сообщений и отладочной информации можно следующими специальными командами:

- `\FPmessagestrue` — разрешает вывод сообщений,
- `\Fpmessagesfalse` — запрещает вывод сообщений,
- `\FPdebugtrue` — разрешает вывод отладочной информации,
- `\FPdebugfalse` — запрещает вывод отладочной информации.

Пакет `fp` работает с числами фиксированной точности (18 знаков до и столько же после запятой) в допустимом диапазоне
от -999999999999999999.999999999999999999
до 999999999999999999.999999999999999999.

Переменные в `fp` указываются двумя основными способами. Во-первых, обозначение переменной может начинаться с символа \(backslash), за которым указывается имя переменной. Это имя должно удовлетворять стандартным требованиям к строковым константам. Например, переменные `\x` и `\omega`. Во-вторых, имя переменной может указываться в фигурных

скобках, например `{omega}`. Выражения, обозначающие операции над переменными и числами или просто числа заключаются в фигурные скобки, например `{-3}` и `{2*z+3}`.

Все команды пакета `fp` начинаются с символа \(backslash) и заглавных букв `FP` без пробелов, т. е. `\FP`.

Из базовых команд пакета рассмотрим те, которые будут непосредственно использоваться в приведенном ниже примере составления контрольных заданий.

В пакете `fp` реализованы функции, позволяющие пользователю работать с выражениями в инфиксной и постфиксной записи.

Для вычисления значения выражения в инфиксной записи используется команда `\FPeval`. Первым аргументом является имя переменной, например `\x`, а вторым — запись вычисляемого выражения, указываемого в фигурных скобках. Допускается использование знаков арифметических операций `+`, `-`, `*`,`/` и знака возведения в степень `^`, всех математических функций пакета, причем в именах функций не нужно использовать символы `FP`.

Например, команда для вычисления выражения $\sqrt{\sin 2.5}$:

```
\FPeval\result{round(root(2,sin(2.5)):2)}
```

присвоит переменной `result` значение 0.77.

В результате выполнения команды

```
\Fpeval\result{clip(2*3+5*6)}
```

переменной `result` будет присвоено значение 36 (без незначащих нулей). Здесь `clip(2*3+5*6)` означает отбрасывание все незначащих нулей результата вычисления выражения `(2*3+5*6)`.

Далее встретится также функция `round`, означающая округление, например, `round(2*x):3` означает округление результата вычисления выражения `2x до трех знаков после запятой`.

Результат вычислений выводится на экран командой `\FPprint`. Команда `\FPprint` предназначена для вывода значения переменной или числа (но не выражения!). Например, `\FPprint\z` (или `\FPprint{z}`) и `\FPprint{2.3}` выведут соответственно числовое значение переменной z

и число 2.3. При этом значение переменной z и число 2.3 будут выведены с 18 цифрами после запятой, поэтому необходимо использовать функции `round` или `clip` для указания требуемой точности вывода числа.

Команда `\FPrandom\x` записывает в переменную x реализацию равномерно распределенной случайной величины на отрезке [0,1].

2.2. Использование пакета fp для генерации заданий

Покажем, как пакет `fp` может быть использован для автоматической подготовки заданий контрольных работ по экономической теории со случайными входными данными (пример А.С. Величко).

Текст основного файла `test.tex` контрольной работы выглядит следующим образом:

```
\documentclass[12pt,a4paper]{article}
\usepackage[cp1251]{inputenc}
\usepackage[russian]{babel}
\usepackage{fp}
\begin{document}     \input{seed}
\begin{center} Контрольная работа \No1\\
Вариант \FPprint{n} \end{center}
\FPrandom\x \FPrandom\y \FPrandom\z
\FPrandom\p \FPrandom\q \FPrandom\r
\FPrandom\xx \FPrandom\yy \FPrandom\zz
\FPrandom\pp \FPrandom\qq \FPrandom\rr
\FPeval\x{round((1+10*x):0)}
\FPeval\y{round((1+10*y):0)}
\FPeval\z{round((1+10*z):0)}
\FPeval\p{round((1+10*p):0)}
\FPeval\q{round((1+10*q):0)}
\FPeval\r{round((1+10*r):0)}
\FPeval\xx{round((1+10*xx):0)}
\FPeval\yy{round((1+10*yy):0)}
\FPeval\zz{round((1+10*zz):0)}
```

```
\FPeval\pp{round((1+10*pp):0)}
\FPeval\qq{round((1+10*qq):0)}
\FPeval\rr{round((1+10*rr):0)}
\FPeval\pp{clip(pp+p)}\FPeval\qq{clip(qq+q)}
\FPeval\rr{clip(rr+r)}
\FPeval\xx{clip(xx+x)}\FPeval\yy{clip(yy+y)}
\FPeval\zz{clip(zz+z)}
```

```
{\bf Задание 3}. В экономике производится только
три товара: апельсины, автомобили и компьютеры. По
данным, приведенным в таблице, рассчитайте
реальный и номинальный ВВП в 1996 г., считая 1995
год базовым, дефлятор ВВП, инфляцию и темп роста
реального ВВП в 1996 г. по отношению к 1995 г.
\begin{table}[!h]
```

```
\begin{center}
\begin{tabular}{|l|c|c|c|c|}
\hline
Годы & \multicolumn{2}{c|}{1995г.} &
\multicolumn{2}{c|}{1996г.} \\
\hline Продукты & Цена & Кол-во & Цена & Кол-во
\\
\hline Апельсины & \FPprint{p} & \FPprint{x} &
\FPprint{pp} & \FPprint{xx} \\
\hline Автомобили & \FPprint{q} & \FPprint{y} &
\FPprint{qq} & \FPprint{yy} \\
\hline Компьютеры & \FPprint{r} & \FPprint{z} &
\FPprint{rr} & \FPprint{zz} \\
\hline
\end{tabular}
\end{center}
\end{table}
\ end{document}
```

Промежуточный файл **seed.tex**, содержит две команды пакета **fp**:

```
\FPset\n{1} \FPseed=1
```

Переменная n, присваиваемая командой **\FPset**, отвечает за номер варианта контрольной работы (в данном случае 1), а команда **\FPseed** инициализирует генератор псевдослучайных чисел значением 1 (это число должно быть различным для разных вариантов контрольных работ, иначе будут созданы одинаковые наборы входных данных!).

Следующий файл **key.tex** содержит текст ответов к контрольному заданию. Полный текст файла:

```
\documentclass[12pt,a4paper]{article}
\usepackage[cp1251]{inputenc}
\usepackage[russian]{babel}
\usepackage{fp}
\begin{document}
\input{seed}
\begin{center} Контрольная работа \No 1 \\
Ответы. Вариант \FPprint{n} \end{center}
\FPrandom\x \FPrandom\y \FPrandom\z
\FPrandom\p \FPrandom\q \FPrandom\r
\FPrandom\xx \FPrandom\yy \FPrandom\zz
\FPrandom\pp \FPrandom\qq \FPrandom\rr
\FPeval\x{round((1+10*x):0)}
\FPeval\y{round((1+10*y):0)}
\FPeval\z{round((1+10*z):0)}
\FPeval\p{round((1+10*p):0)}
\FPeval\q{round((1+10*q):0)}
\FPeval\r{round((1+10*r):0)}
\FPeval\xx{round((1+10*xx):0)}
\FPeval\yy{round((1+10*yy):0)}
```

```
\FPeval\zz{round((1+10*zz):0)}
\FPeval\pp{round((1+10*pp):0)}
\FPeval\qq{round((1+10*qq):0)}
\FPeval\rr{round((1+10*rr):0)}
\FPeval\pp{clip(pp+p)} \FPeval\qq{clip(qq+q)}
\FPeval\rr{clip(rr+r)}
\FPeval\xx{clip(xx+x)} \FPeval\yy{clip(yy+y)}
\FPeval\zz{clip(zz+z)}
\noindent{\bf Задание 3}.
Номинальный ВВП 1995 г. равен
\FPeval\gdpa{clip(p*x+q*y+r*z)} \FPprint{gdpa},
номинальный ВВП 1996 г. равен
\FPeval\gdpb{clip(pp*xx+qq*yy+rr*zz)}
\FPprint{gdpb},
реальный ВВП 1996 г. равен
\FPeval\rgdp{clip(p*xx+q*yy+r*zz)} \FPprint{rgdp},
дефлятор равен
\FPeval\defl{round((gdpb/rgdp*100):1)}
\FPprint{defl}\%,
инфляция равна \FPeval\infl{round(((gdpb/rgdp-
1)*100):1)} \FPprint{infl}\%, темп роста реального
ВВП равен \FPeval\gr{round(((rgdp/gdpa-1)*100):1)}
\FPprint{gr}\%.
\ end{document}
```

Заметим, что в файле ответов `key.tex` дублируются команды, генерирующие исходные данные. Важным для генерации случайных входных данных является тот факт, что использование одного и того же значения для инициализации генератора псевдослучайных чисел (команда `\FPseed`) приводит к одной и той же последовательности псевдослучайных чисел, которые генерируются командами `\FPrandom`. Поэтому последовательность вызовов команд `\FPrandom` должна быть одной и той же для файла с текстом задания и файла с текстом ответов к заданию. Ес-

ли генератор псевдослучайных чисел инициализировать другим значением, например, 3, что соответствует новому варианту контрольной работы, это приведет к генерации нового набора входных данных.

Можно создать текст ответов к заданиям контрольной работы в том же файле `test.tex` с текстом самой работы, например, на новой странице. Во избежание путаницы при генерации случайных данных, желательно использовать уникальные имена переменных, используемых в рамках пакета `fp`, внутри одного файла.

Файл `z_pdf.bat`, содержание которого приведено ниже, предназначен для автоматической генерации нескольких вариантов контрольных работ при работе с издательской системой TEX в операционной системе `Windows`. В строке 1 инициализируется цикл по переменной `i`, которая принимает значения от 1 до 100 с шагом 1 (будет создано 100 контрольных работ). В строках 2-3 создается новый промежуточный файл `seed.tex`, содержащий две команды пакета `fp`:

```
\FPset\n{i} \FPseed=i
```

Вместо переменной `i` подставляется числовое значение, присваиваемое при выполнении цикла. Переменная `n`, присваиваемая командой `\FPset` отвечает за номер варианта контрольной работы, а команда `\FPseed` инициализирует генератор псевдослучайных чисел значением `i` (это число должно быть различным для разных вариантов контрольных работ, иначе будут созданы одинаковые наборы входных данных).

В строках 4 и 6 запускается `pdflatex` для преобразования текста контрольной работы из файла `test.tex` в файл формата `pdf`. Полученный файл `test.pdf` перемещается с помощью команды `move` в файл с уникальным именем, содержащим номер варианта, в данном случае `test_i.pdf`, где вместо переменной `i` подставлено числовое значение, присваиваемое при выполнении цикла.

В строках 5 и 7 аналогичные действия проводятся для файла `key.tex`, который содержит ответы на задания контрольной работы. В строке 8 после выхода из цикла промежуточный файл `seed.tex` удаляется с диска.

Текст файла `z_pdf.bat`:

```
for /L %%i in (1,100,1) do (
echo \FPset\n{%%i} > seed.tex
echo \FPseed=%%i >> seed.tex
pdflatex test.tex
move test.pdf test_%%i.pdf
pdflatex key.tex
move key.pdf key_%%i.pdf )
del /Q seed.tex
```

В результате обработки файла контрольной работы `test.tex` и текста ответов `key.tex` пакетным получатся следующие тексты:

Контрольная работа №1
Вариант 1

Задание 3. В экономике производится только три товара: апельсины, автомобили и компьютеры. По данным, приведенным в таблице, рассчитайте реальный и номинальный ВВП в 1996 г., считая 1995 год базовым, дефлятор ВВП, инфляцию и темп роста реального ВВП в 1996 г. по отношению к 1995 г.

Годы	1995г.		1996г.	
Продукты	Цена	Кол-во	Цена	Кол-во
Апельсины	10	1	16	10
Автомобили	5	8	13	9
Компьютеры	6	8	7	10

Контрольная работа №1
Ответы. Вариант 1

Задание 3. Номинальный ВВП 1995 г. равен 98, номинальный ВВП 1996 г. равен 347, реальный ВВП 1996 г. равен 205, дефлятор равен 169.3%, инфляция равна 69.3%, темп роста реального ВВП равен 109.2%.

Вместо команды `pdflatex` для подготовки файлов формата `pdf` или `ps` можно воспользоваться командами `latex, dvips и dvipdfm`.

Раздел 4. Графика в LaTeX

Лекция 23

METAPOST и TeX

О языке «Meta». Программы METAFONT и METAPOST. Простейшие примеры программ для рисования на языке «Meta». Основные пакеты для расширения возможностей METAPOST'а.

1. METAPOST и TeX

Желание Дональда Э. Кнута добиться качественной печатной продукции привело к созданию не только языка программирования TeX, но и и языка METAFONT. Изначально METAFONT предназначался Д. Кнутом для создания шрифтов и результатом его работы был растр с изображением шрифта. Позднее на основе METAFONT'а Т. Лефрумом (Tom Leathrum) был создан пакет `mfpic`, который используя графические возможности METAFONT'а позволил выполнять довольно качественные рисунки.

Позже аспирант Д.Э.Кнута Джон Хобби (John Hobby) модифицировал METAFONTтаким образом, что результатом работы программы стала картинка в формате `EPS` (`Encusulated PostScript`), добавлена поддержка цвета.

METAPOST проектировался как простая программа, которую можно относительно быстро настроить под свои нужды. Он вполне обозрим и компактен. Многие программы, работающие с векторной графикой, такие как `gnuplot` и `xfig`, умеют экспортировать в METAPOST. Шрифты можно внедрить посредством программ `latex` и `dvips` с результатом в виде `eps`-файла или скрипта `mptopdf` с результатом в виде `pdf`-файла. Эти картинки уже можно использовать независимо любой программой, которая поддерживает эти векторные форматы.

METAPOSTне занимает место средств для ручного рисования или даже интерактивных графических редакторов. Это настоящий язык программирования для генерации графики и, особенно, иллюстраций для документов TEX.

Наши лекции представляют язык METAPOST, начиная с простейших для использования и наиболее важных для простых приложений свойств. Чтение не требует знания METAFONT'а или знакомства с книгой Д. Кнута The METAFONTbook, но обе возможности будут полезными для дальнейшего развития техники рисования.

Домашняя страница METAPOST— http://tug.org/METAPOST. Она содержит ссылки на много дополнительной информации, включая множество статей, которые написаны о METAPOST.

Взаимодействие между TEX'ом и METAPOST'ом может быть двояким. С одной стороны, графика METAPOST может импортироваться в документы, набираемые в TEX'е. С другой стороны, METAPOST'у можно поручить набор текстовых элементов TEX'а, например, текстовых меток или математических формул в графике. Таким способом графика в METAPOST может легко принять стиль документа (шрифт, размер шрифта и т. п.) и соответствовать качеству его набора (использовать кернинг, лигатуры и т. п.). Это делает METAPOST идеальным инструментом для приготовления высококачественной графики для документов в среде TEX.

Мы рассмотрим первую сторону взаимодействия TEX–METAPOST: импорту графики METAPOSTв TEX с использованием пакета `mfpic`.

Для использования METAPOST необходимо приготовить входной файл с METAPOST-кодом и затем вызвать сам METAPOST при помощи, как правило, команды в форме

```
mp <имя файла>.
```

Входные файлы для METAPOST обычно имеют имена, заканчивающиеся на `.mp`, и эта часть имени может опускаться при вызове METAPOST. Например, для входного файла `foo.mp`

```
mp foo
```

вызовет METAPOST и произведет выходные файлы с именами типа `foo.1` и `foo.2`.

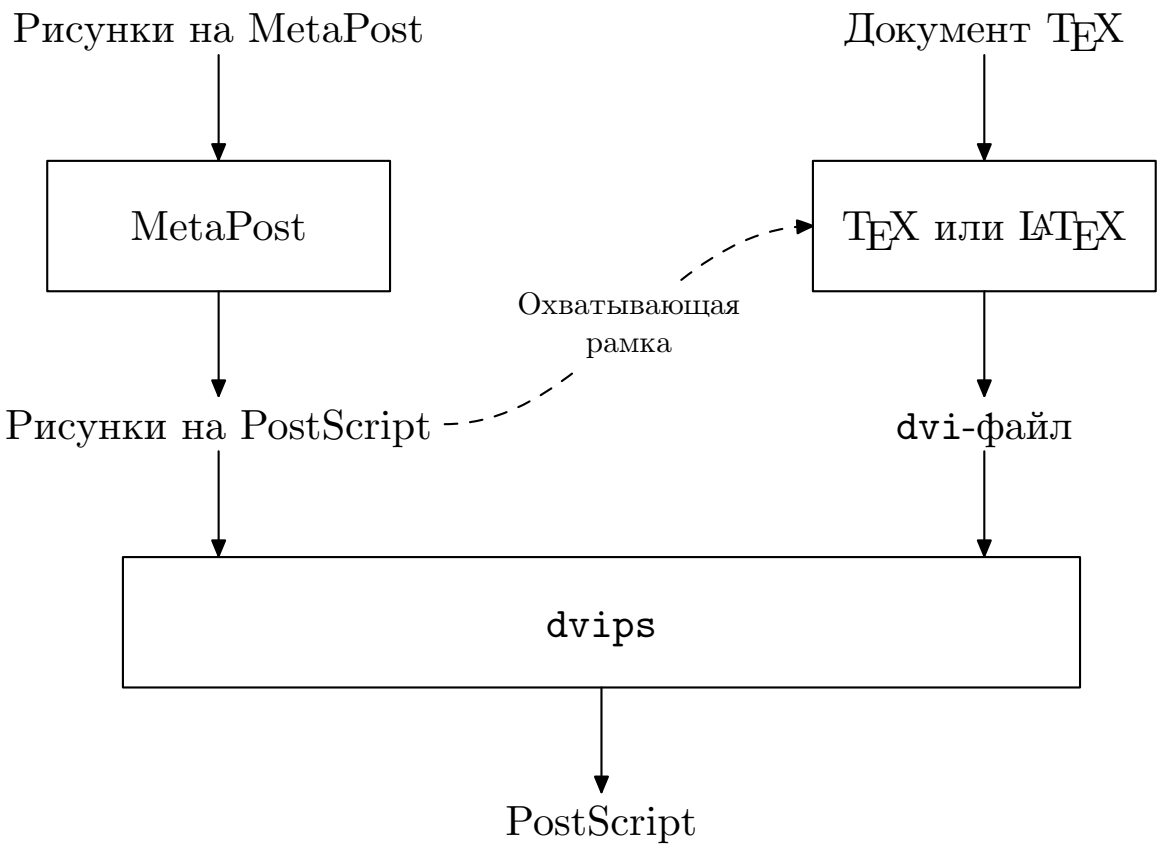


Рис. 23.1. Диаграмма обработки для документа с рисунками METAPOST

Все сообщения, появляющиеся на терминале, собираются в файл-дубликат с именем `foo.log`. Туда же включаются сообщения об ошибках.

То как рисунки `MetaPost` могут быть интегрированы с документами, подготовленными в TEX'е и родственных TEX программах, зависит от формата документов и драйвера вывода. Рис. 23.1 показывает процесс работы для plain TEX, LATEX и свободно доступной программы `dvips`.

Простейший файл для вставки графики, подготовленной METAPOST'ом, в документ TEX'а имеет вид:

```
\documentclass{article}
\usepackage{graphics}
\begin{document}
\includegraphics{foo.1}
\ end{document}
```

С использованием PDF с TEX и LATEX ситуация несколько иная.

В отличие от dvips, pdfLaTeX не может обрабатывать обычные PostScript-файлы — он может работать только с так называемыми очищенными EPS-файлами, которые могут использовать только ограниченное множество возможностей языка PostScript. К счастью, вывод MetaPost — это и есть очищенный EPS, так что тут нам повезло. Из того, что mps — это типовое расширение pdfLaTeX для очищенных EPS-файлов, а вывод MetaPost обычно имеет расширения-числа, мы должны сказать pdfLaTeX обрабатывать занумерованные файлы MetaPost согласно правилам для файлов mps или изменить расширение файла вывода MetaPost на mps. При первом подходе мы должны добавить строку

```
\DeclareGraphicsRule{*}{mps}{*}{}
```

к преамбуле документа после загрузки пакета graphicx. Эта декларация скажет PDFLaTeX загружать все файлы с неизвестным расширением как mps-файлы. См. документацию по пакетам graphicx и graphics для дополнительной информации.

С версии MetaPost 1.000 рекомендован второй подход. Примитив MetaPost filenametemplate может быть использован для установки расширения файла вывода MetaPost в mps. Поэтому декларация \DeclareGraphicsRule здесь не нужна. Более того, расширение может опускаться в команде \includegraphics.

LaTeXи PDFLaTeX. Если вы хотите сохранить гибкость и возможность компилировать как LaTeX, так и PDFLaTeX, то нужно позаботиться о некоторых вещах. Стандартная декларация \DeclareGraphicsRule может быть активирована только, если PDFLaTeX исполняется в PDF-режиме. Поэтому универсальная декларация должна выглядеть подобно этой:

```
\usepackage{graphicx}
\usepackage{ifpdf}
\ifpdf
\DeclareGraphicsRule{*}{mps}{*}{}
\fi
```

Если вы используете метод `filenametemplate`, то расширение файла `mps` не следует опускать в команде `\includegraphics`, т. к. `mps` — это не часть имени LaTeX-файла, заполняемая по умолчанию. Если расширение mps присутствует, то LaTeX обрабатывает эти файлы как `eps`-файлы, что очевидно является корректным. Для дополнительной информации см. описание `\DeclareGraphicsExtensions` и `\DeclareGraphicsRule` в документации пакетов `graphicx` и `graphics`.

2. Кривые

В METAPOST'е основными объектами при рисовании являются кривые и прямые линии. Команда `draw` с разделенными `..` аргументами-точками рисует плавную кривую через эти точки. Например, посмотрите на результат

```
draw z0..z1..z2..z3..z4
```

после определения пяти точек таким образом:

```
z0 = (0,0);   z1 = (60,40);
z2 = (40,90); z3 = (10,70); z4 = (30,50);
```

Ниже показана кривая, проведенная через точки, помеченные от `z0` до `z4`.

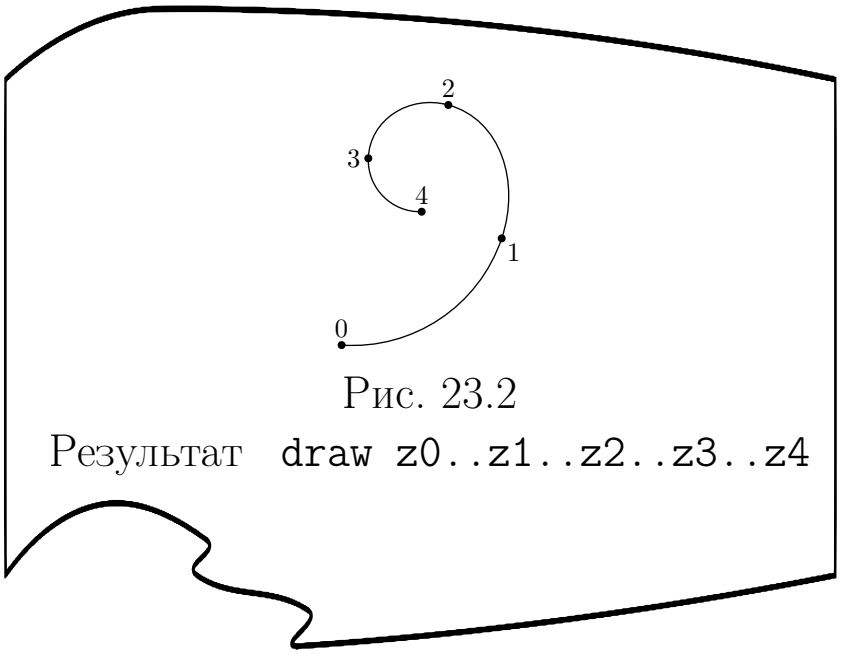


Рис. 23.2
Результат `draw z0..z1..z2..z3..z4`

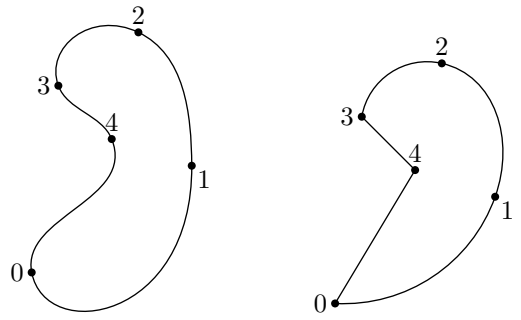
Есть много других способов нарисовать путь через те же самые пять точек.

Для получения гладкой замкнутой кривой соедините `z4` с началом добавлением `..cycle` к команде `draw` как показано на следующем рисунке (случай (a)). Также возможно в одной команде `draw` смешивать кривые и прямые линии как показано на следующем рисунке (кривая (b)). Просто используйте `--` там, где вы хотите прямые линии, и `..` там, где вы хотите кривые. Таким образом,

```
draw z0..z1..z2..z3--z4--cycle
```

произведет кривую через точки 0, 1, 2 и 3, затем ломаную линию из точки 3 в точку 4 и обратно в точку 0. Результат будет точно таким же как после двух команд рисования

```
draw z0..z1..z2..z3
и
draw z3--z4--z0
```



(a) (b)

Рис. 23.3

(a) Результат `draw z0..z1..z2..z3..z4..cycle`;
(b) Результат `draw z0..z1..z2..z3-z4-cycle`

2.1. Кубические кривые Безье

Когда MetaPost просят нарисовать гладкую кривую через последовательность точек, он конструирует кусочную кубическую кривую с непрерывным уклоном и с приблизительно непрерывной кривизной. Это значит, что спецификация пути такая как

```
z0..z1..z2..z3..z4..z5
```

дает в результате кривую, что может быть определена параметрически как $(X(t), Y(t))$ для $0 \leqslant t \leqslant 5$, где $X(t)$ и $Y(t)$ — кусочные кубические функции. Таким образом существуют различные пары кубических функций для каждого ограниченного целыми числами интервала для t. Если z0 $= (x_0, y_0)$, z1 $= (x_1, y_1)$, z2 $= (x_2, y_2)$, ..., то MetaPost выбирает контрольные точки Безье (x_0^+, y_0^+), (x_1^-, y_1^-), (x_1^+, y_1^+), ..., где

$$\begin{aligned} X(t+i) &= (1-t)^3 x_i + 3t(1-t)^2 x_i^+ + 3t^2(1-t) x_{i+1}^- + t^3 x_{i+1}, \\ Y(t+i) &= (1-t)^3 y_i + 3t(1-t)^2 y_i^+ + 3t^2(1-t) y_{i+1}^- + t^3 y_{i+1} \end{aligned}$$

для $0 \leq t \leq 1$. Точные правила для выбора контрольных точек Безье приведены в [?] и в *METAFONTbook* [?].

Для того, чтобы путь имел непрерывный уклон в (x_i, y_i) входящее и исходящее направления в $(X(i), Y(i))$ должны соответствовать. Таким образом, вектора

$$(x_i - x_i^-,\ y_i - y_i^-) \qquad \text{и} \qquad (x_i^+ - x_i,\ y_i^+ - y_i)$$

должны иметь одинаковое направление, т. е. (x_i, y_i) должна быть на отрезке линии между (x_i^-, y_i^-) и (x_i^+, y_i^+). Эта ситуация иллюстрируется на рис. 23.4, где контрольные точки Безье, выбираемые MetaPost, соединены пунктирными линиями. Для тех, кто знаком с интересными свойствами такой конструкции, MetaPost позволяет специфицировать контрольные точки напрямую в следующем формате:

```
draw
    (0,0)..controls (26.8,-1.8) and (51.4,14.6)
 ..(60,40)..controls (67.1,61.0) and (59.8,84.6)
 ..(40,90)..controls (25.4,94.0) and (10.5,84.5)
 ..(10,70)..controls ( 9.6,58.8) and (18.8,49.6)
 ..(30,50);
```

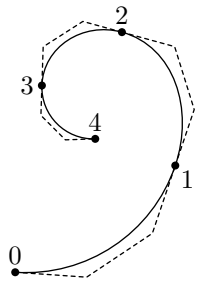


Рис. 23.4
Результат `draw z0..z1..z2..z3..z4` с автоматически выбираемой управляющей ломаной Безье, иллюстрируемой пунктирными линиями.

2.2. Спецификация направления, напряжения и изгиба

`MetaPost` обеспечивает много способов управления поведением пути кривой без действительного указания контрольных точек. Например, некоторые точки на пути могут быть выбраны как вертикальный или горизонтальный экстремумы. Если `z1` следует быть горизонтальным экстремумом, а `z2` — вертикальным, то вы можете указать, что $(X(t), Y(t))$ должна идти вверх в `z1` и влево в `z2`:

```
draw z0..z1{up}..z2{left}..z3..z4;
```

Картинка-результат, показанная на рис. 23.5, имеет желаемые вертикальное и горизонтальное направления в `z1` и `z2`, но она не выглядит такой плавной, как кривая на рис. 23.2. Это обусловлено большим разрывом в величине кривизны в `z1`. Если явно не указать направление в `z1`, то MetaPost-интерпретатор выберет направление таким, чтобы кривизна над `z1` была почти такой же как и кривизна под этой точкой.

Как может выбор направлений в данных точках на кривой определять будет ли кривизна непрерывной?

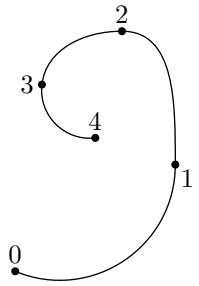
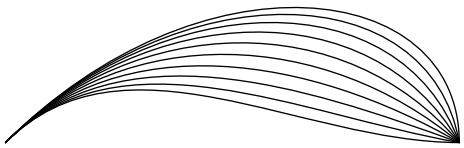


Рис. 23.5
Результат `draw z0..z1{up}..z2{left}..z3..z4`.

Ответ в том, что кривые, используемые в MetaPost, пришли из семейства, где путь определяется своими концами и направлениями там. Рисунки 23.6 и 23.7 дают хорошую идею о том, на что похоже это семейство кривых.



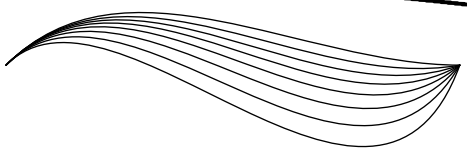


Рис. 23.6 Рис. 23.7
Семейства кривых

Инструкции `MetaPost` для генерации семейств кривых на рис. 23.6 и 23.7 соответственно:

```
beginfig(6)
for a=0 upto 9:
  draw (0,0){dir 45}..{dir -10a}(6cm,0);
endfor
endfig;
```

```
beginfig(7)
for a=0 upto 7:
  draw (0,0){dir 45}..{dir 10a}(6cm,0);
endfor
endfig;
```

Рисунки 23.6 и 23.7 иллюстрируют новые возможности `MetaPost`. Первая — это оператор `dir`, который по углу в градусах генерирует единичный вектор в этом направлении. Таким образом, `dir 0` эквивалентен `right` и `dir 90` эквивалентен `up`. Есть также готовые вектора направлений `left` и `down` для `dir 180` и `dir 270`.

Вектора направлений, заданные в `{}`, могут быть любой длины и они могут как входить в точку, так и выходить из нее. Возможно даже в спецификации пути иметь оба направления для одной точки, до и после. Например, участок спецификации пути

```
        ..{dir 60}(10,0){up}..
```

произведет кривую с углом в $(10, 0)$.

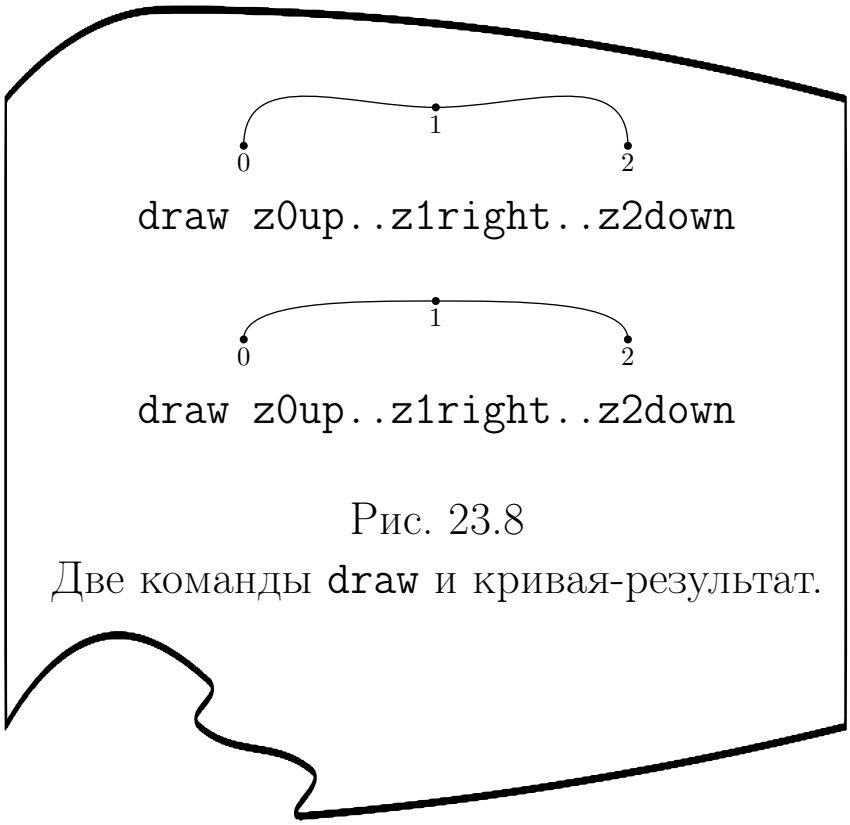


Рис. 23.8
Две команды `draw` и кривая-результат.

Заметьте, что некоторые кривые на рис. 23.6 имеют точки перегиба. Это необходимо при создании гладкой кривой в ситуации подобной рис. 23.3a, но это вероятно нежелательно при работе с вертикальными и горизонтальными точками экстремума как показано на рис. 23.8a. Если `z1` нужно сделать наивысшей точкой на кривой, то это можно получить, используя `...` вместо `..` в спецификации пути как показано на рис. 23.8b. Значение `...` — это “выбрать свободный от перегибов путь между этими точками, если направления в концовых точках дают такую возможность”. На рис. 23.6 возможно избавиться от перегибов, а на рис. 23.7 — нет.

Другой способ управлять неподходящим путем в увеличении параметра "напряжение". Использование `..` в спецификации пути устанавливает параметр напряжения в типовое значение 1. Если это делает некоторую часть пути слишком дикой, то мы можем выборочно увеличить напряжение. Если рис. 23.9a рассматривается как «слишком дикий», то команда `draw` в следующей форме увеличит напряжение между `z1` и `z2`:

```
draw z0..z1..tension 1.3..z2..z3
```

Это произведет рис. 23.9b. Для асимметричного эффекта подобного рис. 23.9c, команда `draw` получает вид

```
draw z0..z1..tension 1.5 and 1..z2..z3
```

Параметр напряжения может быть меньшим единицы, но он должен быть не менее $\frac{3}{4}$.

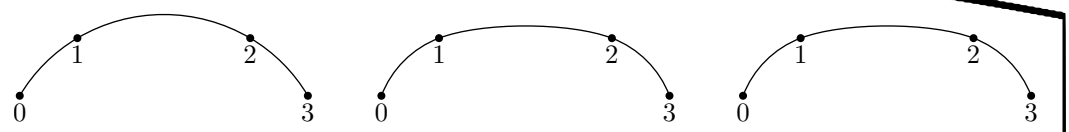
Пути MetaPost имеют также параметр, называемый «изгиб», который влияет на концы путей. При отсутствии спецификаций направления первый и последний отрезки нециклического пути — это приблизительно дуги окружности, как в случае $c = 1$ на рис. 23.10.

Для использования другого значения для параметра изгиба укажите `{curl c}` для некоторого значения c.

Таким образом,

```
draw z0{curl c}..z1..{curl c}z2
```

установит параметр изгиба для `z0` и `z2`.

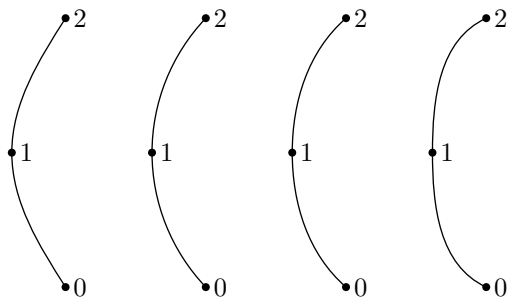


(a) (b) (c)

Рис. 23.9

Результаты `draw z0..z1..tension` α и β `..z2..z3` для разных α и β: (a) $\alpha = \beta = 1$; (b) $\alpha = \beta = 1.3$; (c) $\alpha = 1.5$, $\beta = 1$.

Маленькие значения параметра изгиба уменьшают кривизну в указанных концовых точках пути, а большие значения увеличивают кривизну как показано на рис. 23.10. В частности, значение изгиба ноль делает кривизну нулевой.



c=0 c=1 c=2 c=∞

Рис. 23.10

На рис. 23.10 изображены эффекты изменения параметра изгиба, т. е. результаты `draw z0{curl c}..z1..{curl c}z2` для разных значений параметра изгиба *c*.

Лекция 24

Пакет MFPic

Пакет MFPic как инструмент взаимодействия LaTeX'а и METAPOST'а. Запуск пакета и основные команды для рисования.

В данной лекции мы опишем применение пакета `mfpic` для подготовки качественных рисунков с использованием METAPOST'а, не затрагивая при этом сам язык METAPOST, а используя только команды и инструменты из пакета. Во многих случаях этого вполне хватает. Об использовании средст самого языка программирования METAPOST поговорим позднее.

Первоначально `mfpic` был написан Т. Литрамом (Tom Leathrum) в 1992 году. Он состоял из пакета на языке TeX'а (который можно использовать вместе с LaTeX'ом) и служебной библиотеки макросов на языке METAFONTá. В течение нескольких лет после этого были написаны и проверены различные версии. Многие разработки были выполнены другими авторами.

В 1997-99 годах пакет был дополнен библиотекой макросов на языке METAPOST'а (Dan Luecking). Это позволило, в частности, подключить цвет при рисовании. Мы приводим примеры работы с пакетом в этом последнем варианте.

При работе с пакетом MFPIC нужно начать с того, что в преамбуле входного файла необходимо загрузить дополнительный пакет (пакет `mfpic` с опцией **metapost**):

```
\begin{fmpage}{0.6\textwidth}
\usepackage[metafpost]{mfpic}
```

Далее дать команду на открытие файла для подготовки работы METAPOST'а и закрытие.

```
\opengraphsfile[<имя-файла>]
...  \closegraphsfile
```

Команду закрытия файла можно дать одну в конце файла, а команд открытия несколько либо одна. После начала компиляции исходного файла будет создан файл, например, foo.mp (если открыли файл с именем foo), в котором будут записаны коды программы для METAPOST'а.

Для того чтобы обработать типовой файл, сначала подействуйте LaTeXóм на файл `file.tex`. Когда файл будет обработан, у Вас будет файл, называемый, например `foo.mf`. Это — METAPOSTíй файл. Вы должны подействовать METAPOST'ом на `foo.mf`. Теперь повторно обработайте файл `file.tex` LaTeX'ом. Возникающий в результате DVI файл теперь должен быть полон, и вы можете печатать и просматривать это на Вашем компьютере. Аналогичные шаги необходимо проделать при использовании PdfLaTeX'a.

Эти три шага обработки не всегда необходимы. В частности, если Вы изменяете LaTeX'ий документ без каких-либо изменений в изображениях, тогда нет потребности повторять любой из последних двух шагов.

1. Окружение mfpic

Окружение

```
\begin{mfpic}%
[<xscale>][<yscale>][<xneg>][<xpos>][<yneg>][<ypos>]
...
\ end{mfpic}
```

открывает и закрывает среду, в которой остальные макрокоманды, описанные ниже, будут иметь смысл.

Окружение `\mfpic` также устанавливает локальную систему координат для изображения. Параметры `<xscale>` и `<yscale>` устанавливают единичную длину в системе координат, как кратный размерности `\mfpicunit` (принятое по умолчанию значение которой равно 1 pt).

По крайней мере один параметр масштаба должен быть определен, но если определен только один параметр, то предполагается, что они равны.

Параметры `<xneg>` и `<xpos>` устанавливают нижний и верхний (соответственно) пределы для координатной оси x; аналогично, `<yneg>` и `<ypos>` устанавливают пределы для оси y.

Эти пределы выражаются в относительных единицах, другими словами, фактическая ширина изображения будет (`<xpos>`-`<xneg>`)·`<xscale>` единиц `\mfpicunit`, высота — (`<ypos>`-`<yneg>`)·`<yscale>` единиц

\mfpicunit, и нулевая глубина. Эти масштабы и пределы используются прежде всего для того, чтобы устанавливать для рисунка, содержащего изображение описанное внутри среды.

1.1. Точки, линии, и прямоугольники

```
\pointdef{(name)}(x,y)
```

Эта макрокоманда присваивает символьное имя точке и ее координатам. То есть, если вы хотите часто использовать точку с координатами, например, (1.54553,45.4645), то намного удобнее будет писать просто a, заранее определив точку a таким образом:

```
\point{(<x0>,<y0>),(<x1>,<y1>),...}
```

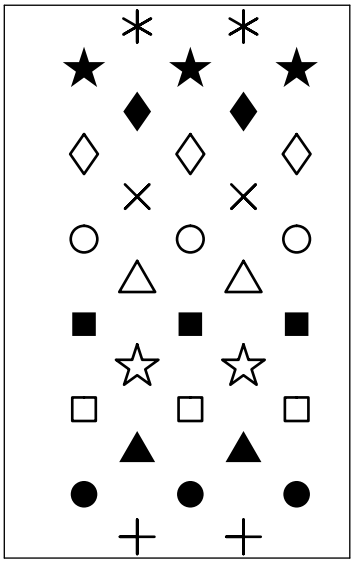
Эта макрокоманда рисует круги с центрами в точках {(<x_0>,<y_0>)}, {(<x_1>,<y_1>)} и т.д. Диаметр окружностей определяется величиной pointsize. Значение pointsize по умолчанию 2 пункта.

```
\plotsymbol[<size>]{<symbol>}{<p0>,<p1>,...}
```

Эта макрокоманде рисует маленькие фигурки в точках <p_0>, <p_1>, и т.д. Этим фигурам должно быть дано имена, такие как: Asterisk, Circle, Diamond, Square, Triangle, Star, SolidCircle, SolidDiamond, SolidSquare, SolidTriangle, SolidStar, Cross и Plus.

```
\begin{mfpic}[10][8]{0}{10}{0}{26}
\plotsymbol{Asterisk}{(3,25),(7,25)}
\plotsymbol{SolidStar}{(1,23),(5,23),(9,23)}
\plotsymbol{SolidDiamond}{(3,21),(7,21)}
\plotsymbol{Diamond}{(1,19),(5,19),(9,19)}
\plotsymbol{Cross}{(3,17),(7,17)}
\plotsymbol{Circle}{(1,15),(5,15),(9,15)}
\plotsymbol{Triangle}{(3,13),(7,13)}
\plotsymbol{SolidSquare}{(1,11),(5,11),(9,11)}
\plotsymbol{Star}{(3,9),(7,9)}
```

```
\plotsymbol{Square}{(1,7),(5,7),(9,7)}
\plotsymbol{SolidTriangle}{(3,5),(7,5)}
\plotsymbol{SolidCircle}{(1,3),(5,3),(9,3)}
\plotsymbol{Plus}{(3,1),(7,1)}
\end{mfpic}
```



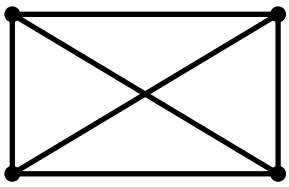
```
\polyline{(<x0>,<y0>),(<x1>,<y1>),...}
\lines{(<x0>,<y0>),(<x1>,<y1>),...}
```

Эти команды идентичные и рисуют отрезок с концами в точках {(<x_0>,<y_0>)} и {(<x_1>,<y_1>)}, затем отрезок с концами в точках {(<x_1>,<y_1>)} и {(<x_2>,<y_2>)}, и т.д. В результате эта макрокоманда нарисует ломанную, соединяя точки в указанном порядке.

```
\rect{(<x0>,<y0>),(<x1>,<y1>)}
```

Эта макрокоманда рисует прямоугольник, определенный точками {(<x_0>,<y_0>)} и {(<x_1>,<y_1>)}, которые являются любыми двумя противоположными вершинами прямоугольника.

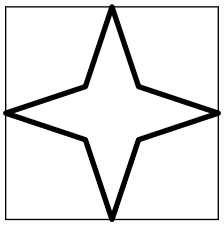
```
\begin{mfpic}[20]{0}{6}{0}{4}
\polyline{(0,0),(5,3)}
\lines{(0,3),(5,0)}
\point[6pt]{(0,0),(5,3),(0,3),(5,0)}
\rect{(0,0),(5,3)}
\end{mfpic}
```



```
\polygon{(<x0>,<y0>),(<x1>,<y1>),...}
```

Эта макрокоманда рисует замкнутый многоугольник с вершинами в заданных точках.

```
\begin{mfpic}[20]{-2}{2}{-2}{2}
\pen{2pt}
\polygon{(-2,0),(-0.5,0.5),(0,2),
(0.5,0.5),(2,0),(0.5,-0.5),(0,-2),
(-0.5,-0.5)}
\end{mfpic}
```



1.2. Оси и метки осей

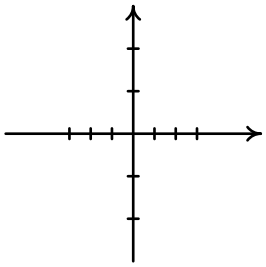
```
\axes[<hlen>]
\xaxis[<hlen>]
\yaxis[<hlen>]
```

Эти макрокоманды рисуют оси x и y для системы координат. Команда `\axes[<hlen>]` идентична командам `\xaxis[<hlen>]` и `\yaxis[<hlen>]`, следующими друг за другом. Параметр <hlen> задает длину стрелки для каждой оси системы координат. Также длина стрелок определяется величиной `\axisheadlen`. Величина `\axisheadlen` по умолчанию 5 пунктов. Форма наконечника стрелки определяется как в макрокоманде `\arrow`.

```
\xmarks[<len>]{<numberlist>}
\tmarks[<len>]{<numberlist>}
\bmarks[<len>]{<numberlist>}
\ymarks[<len>]{<numberlist>}
\lmarks[<len>]{<numberlist>}
\rmarks[<len>]{<numberlist>}
\axismarks{<axis>}[<len>]{<numberlist>}
```

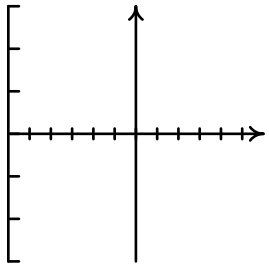
Эти макрокоманды ставят разделяющие метки на соответствующих осях в местах указанных в списке `<numberlist>`. Расстояние между метками определяется величиной `<len>`, и если этот необязательный параметр отсутствует, то расстояние определяется `\hashlen`. Величина `\hashlen` по умолчанию 4 пункта. Команда `\axismarksx` равносильна команде `\xmarks`, а также может использоваться для каждой из шести осей.

```
\begin{mfpic}[8]{-6}{6}{-6}{6}
\axes
\xmarks{-1,-2,-3,1,2,3}
\axismarks{y}{-2,2}
\end{mfpic}
```



Список `<numberlist>`, кроме обычного перечисления координат меток, может содержать цикл, то есть, команда `\xmarks-2 step 1 until 2` будет равносильна команде `\xmarks-2,-1,0,1,2`.

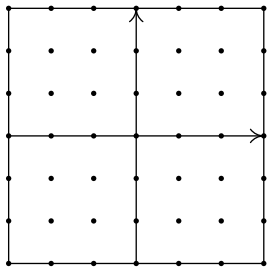
```
\begin{mfpic}[8]{-6}{6}{-6}{6}
\axes
\axis{b}
\xmarks{-5 step 1 until 5}
\lmarks{-6 step 2 until 6}
\end{mfpic}
```



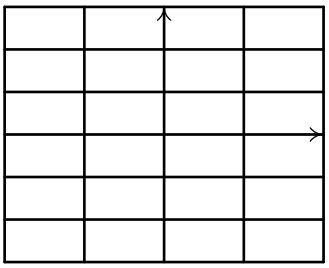
```
\grid[<ptsize>]{<xsep>,<ysep>}
\gridpoints[<ptsize>]{<xsep>,<ysep>}
\lattice[<ptsize>]{<xsep>,<ysep>}
\gridlines{<xsep>,<ysep>}
\plrgrid{<rsep>,<anglesep>}
\plrpatch{<rmin>,<rmax>,<rsep>,<tmin>,<tmax>,<tsep>}
```

Все эти макрокоманды рисуют различные покрытия рисунка. Первые три макрокоманды идентичны и создают точечный шаблон. Плотность точек определяется параметрами `<xsep>` — расстояние между точками по оси , и `<ysep>` — расстояние между точками по оси `y`. Макрокоманда `\gridlines` рисует горизонтальные и вертикальные линии через эти точки.

```
\begin{mfpic}[7]{-6}{6}{-6}{6}
\doaxes{rtbl}
\pen{1pt}
\axes
\grid[2pt]{2,2}
\end{mfpic}
```



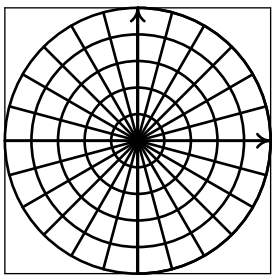
```
\begin{mfpic}[10][8]{-6}{6}{-6}{6}
\doaxes{rtbl}
\axes
\pen{1pt}
\gridlines{1,2}
\end{mfpic}
```

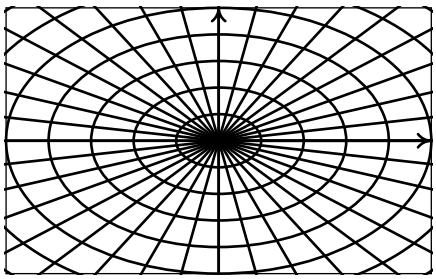


Макрокоманда `\plrgrid` чертит дуги с центром в точке `(0,0)` и линии, исходящие из центра. Соответствующая METAPOST команда действительно рисует такое покрытие площади рисунка, но границы обрезают его. Если вы не хотите, чтобы ваше покрытие было обрезано, то необходимо применять макрокоманду `\plrpatch`, которая рисует дуги начиная с радиуса `<rmin>` шагом `<rsep>` до `<rmax>`, каждая дуга имеет угол от `<tmin>` до `<tmax>`. Также рисуются лучи, начиная с угла `<tmin>` с шагом `<tsep>` и заканчивая углом `<tmax>`.

```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\doaxes{lrbt}
\pen{1pt}
\axes
\plrpatch{-10,10,2,-90,90,15}
\end{mfpic}
```

```
\begin{mfpic}[8][5]{-10}{10}{-10}{10}
\doaxes{lrbt}
\pen{1pt}
\axes
\plrgrid{2,10}
\end{mfpic}
```





1.3. Окружности и эллипсы

```
\circle{(<xc>,<yc>),<r>}
```

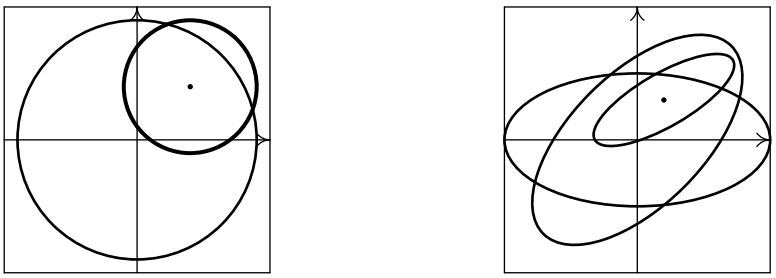
Эта макрокоманда рисует окружность с центром в точке {(<x_c>,<y_c>)} и с радиусом <r>.

```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\doaxes{lrbt}
\axes \pen{1pt}
\circle{(0,0),9}
\pen{1.5pt}\point{(4,4)}
\circle{(4,4),5}
\end{mfpic}
```

```
\ellipse[<\Theta>]{(<x_c>,<y_c>),<r_x>,<r_y>}
```

Эта макрокоманда рисует эллипс с радиусом <r_x> по x и с радиусом <r_y> по y, с центром в точке (<x_c>,<y_c>). Параметр <Θ> задает поворот эллипса, измеряемый в градусах против часовой стрелки вокруг своего центра.

```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\doaxes{lrbt}
\axes \pen{1pt}
\ellipse{(0,0),10,5}
\ellipse[45]{(0,0),10,5}
\point{(2,3)}
\ellipse[30]{(2,3),6,2}
\end{mfpic}
```



2. Гистограмма и секторная диаграмма

```
\barchart<[start>,<sep>,<r>]{<h-or-v>}{<list>}
\chartbar{<num>}
```

Макрокоманда `\barchart` сама ничего не рисует, а создает соответствующие прямоугольники для диаграммы макрокоманда `\chartbar`. Параметр `<h-or-v>` может быть `<v>`, если вы хотите построить вертикальную диаграмму вдоль оси x, или `<h>`, если вы хотите построить горизонтальную вдоль оси y. `<list>` состоит из координат вершины каждого столбика, перечисленных через запятую.

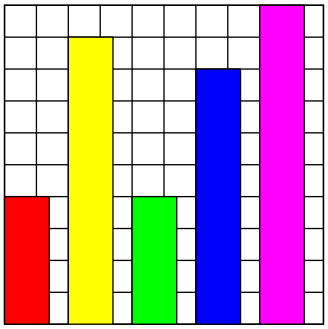
По умолчанию ширина столбиков составляет одну графическую единицу, и ширина n-го столбика простирается в интервале [n-1; n]. Необязательный параметр состоит из трех числовых параметров, разделенных запятыми. `<start>` — координата начала первого столбика (считает от нуля, как для горизонтальной, так и для вертикальной диаграммы), параметр `<sep>` задает расстояние между левыми границами соседних столбиков, и параметр `<r>` составляет часть от `<sep>`, и вычисляется таким

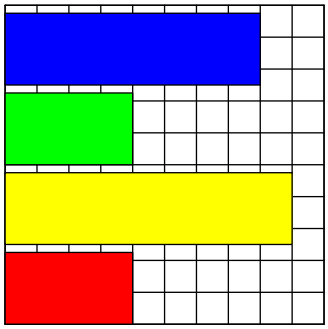
образом, чтобы произведение <r>*<sep> — и было шириной столбцов. По умолчанию соответствует [0,1,1].

Параметр <r> изменяет от -1 до 1. Так как начальная координата диаграммы, т.е. первого столбца, вычисляется по формуле <start>+<r>*<sep>, то отрицательное значение <r> приведет к сдвигу всей диаграммы относительно <start> влево на расстояние <r>*<sep>.

После макрокоманды \barchart должны быть макрокоманды \chartbar[1], \chartbar[2], рисующие сами столбики, и так далее. \chartbar рисует прямоугольник с порядковым номером <num>, который может быть закрашен цветом, заштрихован и так далее, используя соответствующие префиксные макрокоманды (смотрите *Префиксные макрокоманды*).

```
\begin{mfpic}[10]{0}{20}{0}{20}
\barchart[0,2,0.7]{v}{4,9,4,8,10}
\draw\gfill[(1,0,0)]\chartbar{1}
\draw\gfill[(1,1,0)]\chartbar{2}
\draw\gfill[(0,1,0)]\chartbar{3}
\draw\gfill[(0,0,1)]\chartbar{4}
\draw\gfill[(1,0,1)]\chartbar{5}
\end{mfpic}
```





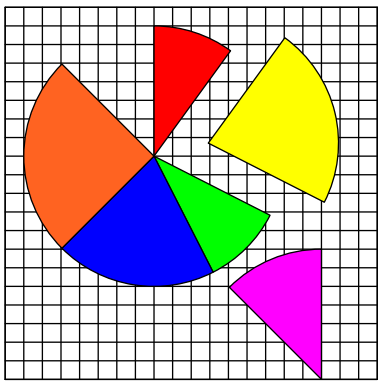
```
\begin{mfpic}[10]{0}{20}{0}{20}
\barchart[0,2.5,0.9]{h}{4,9,4,8,10}
\draw\gfill[(1,0,0)]\chartbar{1}
\draw\gfill[(1,1,0)]\chartbar{2}
\draw\gfill[(0,1,0)]\chartbar{3}
\draw\gfill[(0,0,1)]\chartbar{4}
\end{mfpic}
```

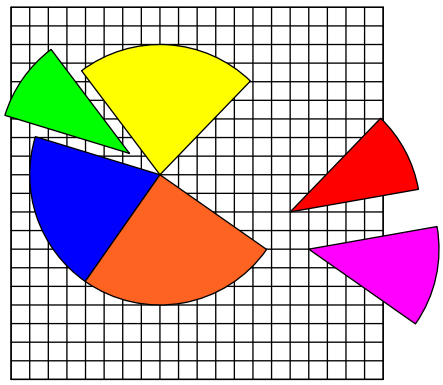
```
\piechart[<dir><angle>]{<c>,<r>}{<list>}
\piewedge[<spec><trans>]{<num>}
```

Макрокоманда `\piechart` тоже ничего сама не рисует. Сектора диаграмы создает макрокоманда `\piewedge`, описанная ниже. Параметр `<dir>` состоит из одной из букв `c` или `a`, которые указывают направление расположения секторов. Буква `c` говорит о том, что сектора будут располагаться по часовой стрелке, `a` — против часовой. По умолчания первый сектор диаграммы начинается с положения стрелки 12 часов дня (угол 90 градусов) и последующие сектора располагаются по часовой стрелке. Параметром `<dir>` вы можете изменить направление стрелки, а параметром `<angle>` угол, с которого начнут располагаться сектора по порядку. Параметр `<c>` задает центр диаграммы с радиусом `<r>`.

После макрокоманды `\piechart` должны использоваться макрокоманды `\piewedge`[1], `\piewedge`[2], и так далее. Параметр `<spec>` состоит из одной буквы `x`, `s` или `m`. Буква `x` означает, что сектор передвинется в направлении от центра диаграммы на расстояние `<trans>`. Буква `s` передвинет сектор на заданное расстояние от его начального положения. В это случае параметр `<trans>` будет парой чисел (`<dx>`,`<dy>`), где `<dx>` — расстояние перемещения по горизонтали, `<dy>` — по вертикали (измеряется в графических единицах). Буква `m` переместит сектор в заданную точку вашего рисунка. Координаты точки задаются параметром `<trans>`. Сектора могут быть закрашены цветом, заштрихованы и так далее, используя соответствующие префиксные макрокоманды (смотрите *Префиксные макрокоманды*).

```
\begin{mfpic}[7]{-10}{10}{-10}{10}
\piechart{(-2,2),7}{4,9,4,8,10,5}
\draw\gfill[(1,0,0)]\piewedge{1}
\draw\gfill[(1,1,0)]\piewedge[x3]{2}
\draw\gfill[(0,1,0)]\piewedge{3}
\draw\gfill[(0,0,1)]\piewedge{4}
\draw\gfill[Orange]\piewedge{5}
\draw\gfill[(1,0,1)]\piewedge[s(9,-12)]{6}
\end{mfpic}
```





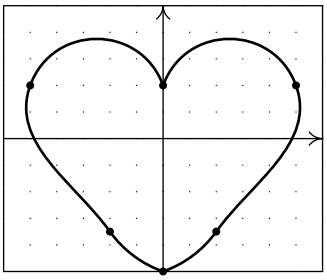
```
\begin{mfpic}[7]{-10}{10}{-10}{10}
\piechart[a10]{(-2,1 ),7}{4,9,4,8,10,5}
\draw\gfill[(1,0,0)]\piewedge[s(7,-2)]{1}
\draw\gfill[(1,1,0)]\piewedge{2}
\draw\gfill[(0,1,0)]\piewedge[x2]{3}
\draw\gfill[(0,0,1)]\piewedge{4}
\draw\gfill[Orange]\piewedge{5}
\draw\gfill[(1,0,1)]\piewedge[m(6,-3)]{6}
\end{mfpic}
```

2.1. Кривые линии

```
\curve[<tension>]{(<x0>,<y0>),(<x1>,<y1>),...}
```

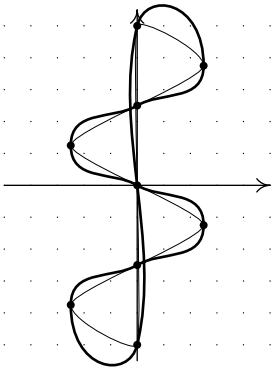
Эта макрокоманда рисует кривую, соединяющую точки в указанном порядке. Необязательный параметр `<tension>` по умолчанию равен 1 и определяет гладкость кривой (изменяется от 1 до 10). Чем больше значение параметра `<tension>`, тем острее углы кривой.

```
\begin{mfpic}[5]{-12}{12}{-10}{10}
\axes \grid{2,2}\pen{1pt}
\point{(0,4),(-10,4),(0,-10)}
\point{(10,4),(-4,-7),(4,-7)}
\curve{(0,4),(-10,4),(-4,-7),(0,-10)}
\curve{(0,4),(10,4),(4,-7),(0,-10)}
\end{mfpic}
```



```
\cyclic[<tension>]{(<x0>,<y0>),(<x1>,<y1>),...}
```

Эта макрокоманда рисует замкнутую кривую, проходящую через точки в указанном порядке.



```
\begin{mfpic}[5][6]{-10}{10}{-11}{11}
\gridpoints{2,2}\axes \pen{1pt}
\cyclic{(0,-10),(-5,-7.5),(0,-5),
(5,-2.5),(0,0),(-5,2.5),(0,5),(5,7.5),(0,10)}
\pen{0.2}
\cyclic[4]{(0,-10),(-5,-7.5),(0,-5),(5,-2.5),
(0,0),(-5,2.5),(0,5),(5,7.5),(0,10)}
\end{mfpic}
```

2.2. Круглые дуги

```
arc[<format>]{...}
```

Эта макрокоманда рисует дугу, определенную параметром `<format>`. Это необычная макрокоманда, так как в ней параметр `<format>` определяет формат другого параметра, как указано ниже:

```
arc[s]{(<x0>,<y0>),(<x1>,<y1>),<sweep>}
```

(*Форма-отметка перемещается — это формат по умолчанию.*) Эта макрокоманда рисует дугу из отметки {(<x0>,<y0>)} в отметку {(<x1>,<y1>)}, при этом проходя угол `<sweep>` градусов. Угол измеряется против часовой стрелки вокруг центра окружности. Если, например, отметки {(<x0>,<y0>)} и {(<x1>,<y1>)} лежат на горизонтальной линии с отметкой {(<x0>,<y0>)} лежащей левее, и `<sweep>` лежит между 0 и 180 (градусов), тогда центр окружности будет выше горизонтальной линии (для того, чтобы угол измерялся против часовой стрелки). Отрицательная величина `<sweep>` дает дугу в другом направление.

```
arc[t]{(<x0>,<y0>),(<x1>,<y1>),(<x2>,<y2>)}
```

(*Форма с тремя отметками.*) Эта макрокоманда рисует дугу, проходящую через три данные отметки.

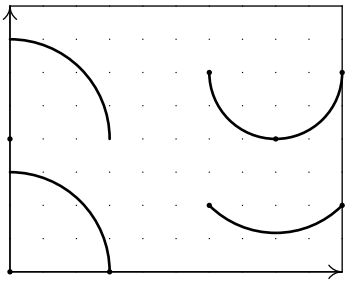
```
arc[p]{(<x_c>,<y_c>),<\theta_1>,<\theta_1>,<r>}
```

(Полярная форма.) Эта макрокоманда рисует дугу с центром (<x_c>,<y_c>) радиуса `<r>`, от угла <θ_1> к углу <θ_2>, где оба угла измеряются против часовой стрелки от положительной оси x.

```
arc[c]{(<x_c>,<y_c>),(<x1>,<y1>),<\theta>}
```

(*Форма средней отметки.*) Это макрокоманда рисует дугу с центром (<x_c>,<y_c>), начиная в отметки (<x_1>,<y_1>) проходя угол <θ> вокруг центра из этой отметки. (Это основной способ задания дуг в `mfpic` — все другие форматы переводятся в этот формат до рисования.)

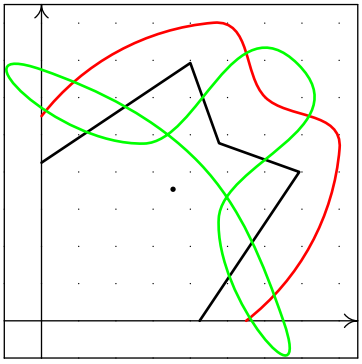
```
\begin{mfpic}[25]{0}{5}{0}{4}
\arc[c]{(0,0),(1.5,0),90}%
\arc[p]{(0,2),0,90,1.5}%
\arc[t]{(3,3),(4,2),(5,3)}%
\arc{(3,1),(5,1),90}
```



2.3. Полярные координаты

```
\point{plr{(<length0 >,<\theta0>),
           (<length1 >,<\theta1>,),...}}
\lines{plr{(<length0 >,<\theta0>),
           (<length1 >,<\theta1>),...}}
\curve{plr{(<length0 >,<\theta0>),
           (<length1 >,<\theta1>),...}}
\cyclic{plr{(<length0 >,<\theta0>),
           (<length1 >,<\theta1>),...}}
```

Эти макрокоманды равносильны `\point`, `\lines`, `\curve` и `\cyclic` соответственно, за исключением того, что все точки выражаются в полярных координатах.



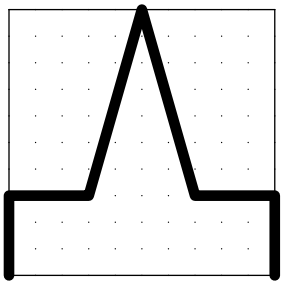
```
\begin{mfpic}[70]{-0.2}{1.7}{-0.2}{1.7}
\point{\plr{(1,45)}}
\lines{\plr{(0.85,0),(1.6,30),
(1.35,45),(1.6,60),(0.85,90)}}%синяя
\curve{\plr{(1.1,0), (1.85,30),
(1.7,45),(1.85,60), (1.1,90)}}%красная
\cyclic{\plr{(1.3,0),(1.1,30),(1.95,45),
(1.1,60),(1.35,90),(1.2,45)}}%зеленая
\end{mfpic}
```

2.4. Другие фигуры

```
\turtle{(<x_c>,<y_c>)(<x1>,<y1>),(<x2>,<y2>),...}
```

Эта макрокоманда рисует отрезок, начинающийся в точке $\{(<x_c>,<y_c>)\}$ и проходящий в направлении, определенном векторами $(<x_1>,<y_1>)$, $(<x_2>,<y_2>)$, ...

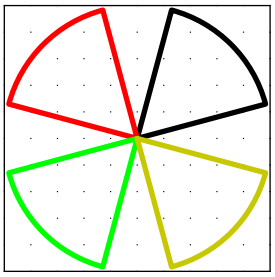
```
\begin{mfpic}[10]{0}{10}{0}{10}
\doaxes{lrbt}
\grid{1,1}\pen{4pt}
\turtle{(0,0),(0,3),(3,0),(2,7),
(2,-7),(3,0),(0,-3)}
\end{mfpic}
```



```
\sector{(<x_c>,<y_c>),<r>,<\theta1>,<\theta2>}
```

Эта макрокоманда рисует клинообразную область от угла $<\theta_1>$ к углу $<\theta_2>$ внутри окружности с центром в точке $(<x_c>,<y_c>)$ радиуса `<r>`, где оба угла измеряются в градусах против часовой стрелки от направленной параллели к оси **x**.

```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\sector{(0,0),10,15,75}%синий
\sector{(0,0),10,105,165}%красный
\sector{(0,0),10,195,255}%зеленый
\sector{(0,0),10,285,345}%желтый
\end{mfpic}
```



3. Префиксные макрокоманды

Некоторые макрокоманды `mfpic` используются как префиксные — например, если вы хотите поместить острие стрелки на отрезок, вы можете написать: `\arrowlines`{(0,0),(1,0)}. Префиксные макрокоманды применяются только к одиночному рисунку. Все префиксные макрокоманды описываются здесь. Для этих макрокоманд в макрокомандах рисунка должно быть сделано различие между «открытым» и «замкнутым» рисунками. Замкнутые фигуры: `\rect`, `\circle`, `\ellipse`, `\cyclic`, и `\btwnfcn` (ниже).

3.1. Макро для рисования

```
\draw...
```

Эта макрокоманда восстанавливает прорисовку контура фигуры, когда он отменяется каким-либо из префиксов заполнения области: `\shade` или `\hatch` (смотрите ниже). Эта макрокоманда также имеет смысл без оттеняющих макрокоманд, но не необходима; однако, если это происходит в контексте оттеняющейся макрокоманды, то оттеняющая макрокоманда должна быть в начале. Команда `\drawcolor` определяет цвет.

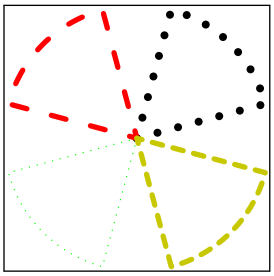
```
\dashed[<length>,<space>]...
```

Эта макрокоманда выводит штриховой сегменты определенного рисунка. Необязательный параметр <length> определяет длину штриховой линии (по умолчанию 4 pt), задается командой \dashlen. Необязательный параметр <space> определяет расстояние между линиями (по умолчанию 4 pt), задается командой \dashspace. Эта макрокоманда корректирует пробел между штриховыми линиями на $\frac{dashspace}{n}$, где n — число пробелов на кривой, для того, чтобы на концах не получились кусочки штриховки. Штриховая линия на концах равна половине длины dashlen. Если эта макрокоманда происходит в контексте оттеняющей макрокоманды, то оттеняющая макрокоманда должна быть в начале. Если эта макрокоманда появляется вместе с \draw, то первый по порядку отменяет другой.

```
\dotted[<size>,<space>]...
```

Эта макрокоманда выводит точеные сегменты определенного рисунка. Необязательный параметр <size> определяет размер точек (по умолчанию 0.5 pt), задается командой \dotsize. Необязательный параметр <space> определяет расстояние между точками(по умолчанию 3 pt), задается командой \dotspace.

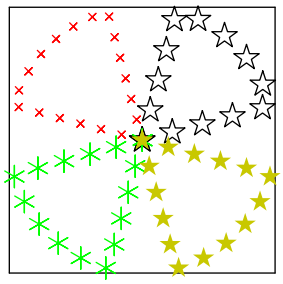
```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\dotted[3pt,8pt]
\sector{(0,0),10,15,75}%синий
\dashed[6pt,10pt]
\sector{(0,0),10,105,165}%красный
\dotted\sector{(0,0),10,195,255}%зеленый
\dashed\sector{(0,0),10,285,345}%желтый
\end{mfpic}
```



```
\plot[<size>,<space>]{<symbol>}...
```

Эта макрокоманда идентична \dotted, за исключением того, что вместо точек рисуются фигурки. Необязательныму параметру <symbol> должна соответствовать одна из фигур: Asterisk, Circle, Diamond, Square, Triangle, Star, SolidCircle, SolidDiamond, SolidSquare, SolidTriangle, SolidStar, Cross и Plus. Размер символа <size> определяется командой \pointsize. Расстояние между символами <space> определяется \symbolspace (по умолчанию 5 pt).

```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\plot[6pt,12pt]{Star}
\sector{(0,0),10,15,75}%синий
\plot[3pt,8pt]{Cross}
\sector{(0,0),10,105,165}%красный
\plot[7pt,10pt]{Asterisk}
\sector{(0,0),10,195,255}%зеленый
\plot[5pt,10pt]{SolidStar}
\sector{(0,0),10,285,345}%желтый
\end{mfpic}
```



```
\plotnodes[<size>]{<symbol>}...
```

Эта макрокоманда рисует сиволы <symbol> размером <size> на границе рисунка, если это \ellips или \circle, и если эта команда стоит перед \lines, \polyline или \curve, то в каждой точке в списке этой команды. Параметр <size> определяется командой \pointsize.

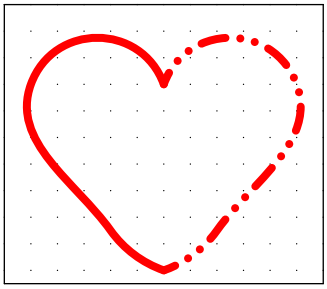
```
\dashpattern{<name>}{<len1 ><len3 >,...,<len2k>}
```

Эта макрокоманда необходима для создания более общего шаблона для штриховки. Она устанавливает имя <name> этого шаблона. Параметры <len1>: : : <len2k> задают длину каждой штриховой линии и расстояние между штриховыми линиями по порядку, и если эта длина равна 0 pt, то будет рисоваться точка.

```
\gendashed{<name>}
```

Если вы заранее определили шаблон для штриховки с именем <name>, то эта команда используется непосредственно перед командой рисования для вызова этого шаблона.

```
\begin{mfpic}[5]{-12}{12}{-10}{10}
\dashpattern{sht1}{10pt,3pt,10pt,3pt,0pt,3pt}
\dashpattern{sht2}{10pt,6pt,0pt,6pt,0pt,6pt}
\gendashed{sht1}
\curve{(0,4),(-10,4),(-4,-7),(0,-10)}
\gendashed{sht2}
\curve{(0,4),(10,4),(4,-7),(0,-10)}
\end{mfpic}
```



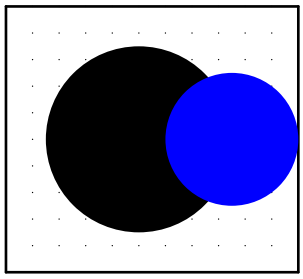
3.2. Макро для заливки, штриховки и стирания рисунка

Эта макрокоманда заполняет цветом <color> замкнутую фигуру. Цвет определяется командой \fillcolor. Эта макрокоманда одновременно отменяет прорисовку контура замкнутой фигуры. Имеет смысл только для замкнутых графических объектов. Если этот префикс указан в списке префиксов, он должен быть первым.

```
\gclear...
```

Эта макрокоманда аналогична `gfill`, но заполнение производится цветом, определенным командой `\backgroundcolor`, и если она отсутствует белым цветом (объекты внутри замкнутой фигуры исчезают).

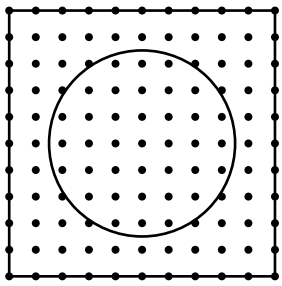
```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\doaxes{lrbt}\grid{2,2}
\backgroundcolor[rgb]{0,0,2}%синий
\gfill\circle{(0,0),7}
\gclear\circle{(7,0),5}
\end{mfpic}
```

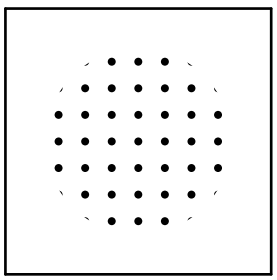


```
\gclip...
```

Эта макрокоманда стирает все за фигурой перед которой стоит.

```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\pen{1pt}
\grid[3pt]{2,2} \backgroundcolor[rgb]{0,0,2}
\circle{(0,0),7} \doaxes{lrbt}
\end{mfpic}
```





```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\grid[3pt]{2,2}
\backgroundcolor[rgb]{0,0,2}
\gclip\circle{(0,0),7}
\doaxes{lrbt}
\end{mfpic}
```

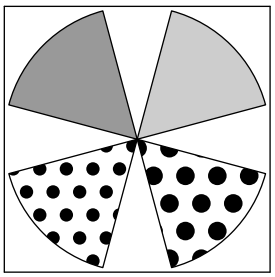
```
\shade[<shadesp>]...
```

Эта макрокоманда производит заполнение замкнутой фигуры серым цветом. Насыщенность цвета определяется расстоянием между точками. Параметр `<shadesp>` — по умолчанию задается размерностью `\shadespace`, и изменяется от 0.5 pt (если меньше 0.5 pt, то фигура будет заполняться черным цветом).

```
\polkadot[<space>]...
```

Эта макрокоманда заполняет замкнутую фигуру точками. Макрокоманды `\shade` и `\polkadot` отличаются друг от друга, последняя команда имеет только черный цвет. Команда `\shade` заполняет замкнутую фигуру точками любых цветов. Параметр <space> задается командой `\polkadotspace` (по умолчанию 10 pt). Диаметр точек задается командой `\polkadotwd<size>` (по умолчанию 5 pt). Точки закрашиваются цветом, определенным `\fillcolor`.

```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\draw\shade\sector{(0,0),10,15,75}
\shadespace=0.7pt
\draw\shade\sector{(0,0),10,105,165}
\shadespace = 1.2pt
\draw\polkadot[10pt]\sector{(0,0),10,195,255}
\polkadotwd{7pt}
\draw\polkadot[12pt]\sector{(0,0),10,285,345}
\end{mfpic}
```



```
\thatch[<hatchsp>,<angle>][<color>]...
```

Эта макрокоманда производит заполнение замкнутой фигуры параллельными линиями под углом `<angle>`. Пробел между линиями определяется необязательным параметром `<hatchsp>` — по умолчанию дается размерностью `\hatchspace`. Цвет линий задается командой `\hatchcolor`

```
\lhatch[<hatchsp>][<color>]...
```

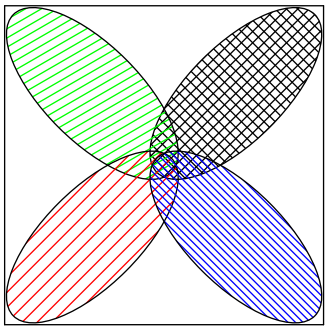
Эта макрокоманда производит заполнение замкнутой фигуры параллельными линиями под углом -45. Тоже самое, что и `\lhatch[<-45>][<color>]...`

```
\rhatch[<hatchsp>][<color>]...
```

Эта макрокоманда производит заполнение замкнутой фигуры параллельными линиями под углом 45. Тоже самое, что и `\lhatch[<45>][<color>]...`

```
\hatch[<hatchsp>]...
\xhatch[<hatchsp>]...
```

Эта макрокоманда производит заполнение замкнутой фигуры пересекающимися линиями.



```
\begin{mfpic}[5]{-10}{10}{-10}{10}\draw
\xhatch\ellipse[45]{(4.5,4.5),7,3}\draw
\thatch[3pt,30][Green]
\ellipse[315]{(-4.5,4.5),7,3}\draw
\rhatch[4pt][Red]
\ellipse[45]{(-4.5,-4.5),7,3}\draw
\lhatch[2pt][Blue]
\ellipse[315]{(4.5,-4.5),7,3}
\end{mfpic}
```

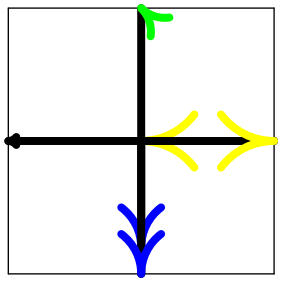
3.3. Макро для изменения стрелок осей координат

```
\arrow[l<headlen>][r<rotate>][b<backset>][c<color>]...
```

Эта макрокоманда рисует острие стрелки в конечной точке открытой фигуры (или в последнем пункте замкнутой фигуры). Необязательный параметр `<headlen>` определяет длину острия. Команда `\headlen`, задающая это значение по умолчанию равна 3pt. Необязательный параметр `<rotate>` задает поворот острия на `<rotate>` градусов против часовой стрелки относительно номинального положения. По умолчанию используется значение 0 градусов. Необязательный параметр `<backset>` смещает положение стрелки назад на заданной расстояние расстояние относительно ее номинального положения. По умолчанию используется значение 0 пункт. Изменение угла наклона стрелки сказывается одновременно на направление, вдоль которого происходит смещение стрелки. Параметр цвета `<color>` определен `\headcolor`. Необязательные параметры могут появляться в любом порядке, но обозначенный символ клавиши (ключ) для каждого параметра должен всегда быть. Если эта макрокоманда происходит в контексте оттеняющей макрокоманды, то оттеняющая макрокоманда должна быть в начале.

```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\arrow[l10pt][r45][cGreen]\lines{(0,0),(0,10)}
```

```
\arrow[cBlue][l15pt][b10pt]\lines{(0,0),(0,-10)}
\arrow[cBlue][l15pt]\lines{(0,0),(0,-10)}
\arrow[cYellow][l20pt][b -50pt][r180]
\lines{(0,0),(10,0)}
\arrow[cYellow][l20pt]\lines{(0,0),(10,0)}
\arrow\lines{(0,0),(-10,0)}
\end{mfpic}
```



3.4. Макро для замыкания разомкнутой фигуры

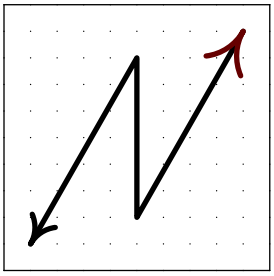
Макрокоманда `\lclosed` замыкает разомкнутую фигуру за счет соединения начальной и конечной точек прямой линией. Если эта макрокоманда происходит в контексте оттеняющейся макрокоманды, то оттеняющаяся макрокоманда должна быть в начале. Эта макрокоманда также содержит подразумеваемую функцию, которая условно ведет себя подобно команде `\draw`, если фигура не штрихуется, эта макрокоманда отменяет любое последующее использование `\dotted`. Макрокоманды `\bclosed`, `\sclosed` и `\cbclosed` подобны `\lclosed`, за исключением того, что они замыкают разомкнутую фигуру кривой, или гладкой кривой (как в гладком случае в макрокоманде `\curve`), или кубическим B-сплайном, соответственно, между конечными точками рисунка.

3.5. Макро для связывания траектории и изменении ориентации рисунка

Макрокоманда `\reverse` меняет местами начальную и конечную точки фигуры, меняет ориентацию. Это повлияет на направление стрелок (т.е. в сочетании с `\arrow` может рисовать две стрелки на концах фигуры:

```
\arrow\reverse
```

\arrow... — здесь первый префикс \arrow относится к повернутой фигуре) и порядок конечных точек для окружения \medskip \connect...\endconnect (ниже). Если эта макрокоманда происходит в контексте оттеняющейся макрокоманды, то оттеняющая макрокоманда должна быть в начале. Эта макрокоманда также содержит подразумеваемую функцию, которая условно ведет себя подобно команде \draw, поэтому если рисунок не заштрихованный, то эта макрокоманда отменяет любое последующее использование \dotted.



```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\doaxes{rltb}\pen{2pt}\arrow[l10pt]
\reverse\arrow[l15pt][cBrown]
\lines{(-8,-8),(0,6),(0,-6),(8,8)}
\end{mfpic}
```

```
\connect...\endconnect
```

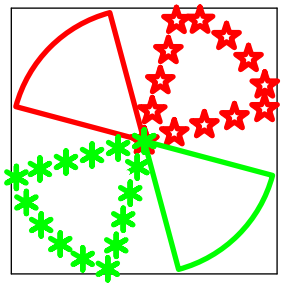
Эти макрокоманды действуют как окружения. Внутри окружения конечная точка предыдущей фигуры соединяется прямой с начальной точкой последующей фигуры. Результатом является открытая непрерывная фигура. Макрокоманда \point ведет себя немного по-другому внутри этого окружения — не рисует точек сама по себе, а только определяет точки, соединяемые ломанной линией. Если это окружение происходит в контексте оттеняющейся макрокоманды, то оттеняющаяся макрокоманда должна быть в начале. Эта макрокоманда также содержит подразумеваемую функцию, которая условно ведет себя подобно команде \draw, поэтому если рисунок не заштрихованный, то эта макрокоманда отменяет любое использование \dotted внутри окружения.

(Это, строго говоря, префиксная команда, но поведение так близко связано с `\lclosed` и `\reverse`, что описывается здесь наряду с ними.)

Примечание: Эта пара команд может задаваться так-же как окружение.

```
\rotatepath{(<x>,<y>),<\theta>}...
```

Эта макрокоманда вращает фигуру относительно точки (`<x>,<y>`) на $<\theta>$ градусов против часовой стрелки.



```
\begin{mfpic}[5]{-10}{10}{-10}{10}
\drawcolor{rgb(2,0,0)}%красный
\rotatepath{(0,0),90}\plot[6pt,12pt]{Star}
\sector{(0,0),10,15,75}
\drawcolor{rgb(0,2,0)}%зеленый
\rotatepath{(0,0),90}\plot[7pt,10pt]{Asterisk}
\sector{(0,0),10,195,255}
\end{mfpic}
```

4. Аффинное (линейное) преобразование координатной системы

Координатные преобразования, которые сохраняют параллельные линии параллельными, называются аффинными преобразованиями. Они включают трансляцию, вращение(циклический сдвиг), отражение сигнала, масштабирование, ассиметрирование. Для METAPOST'ой координатной системы — т.е. для рисунков, а не для меток (исключением являются заголовки) — `mfpic` снабжен способностью применять аффинные преобразования. Преобразования могут быть локализованы в любую группу METAPOST'их маршрутов (рисунков).

```
\begin{coords}...end{coords}
```

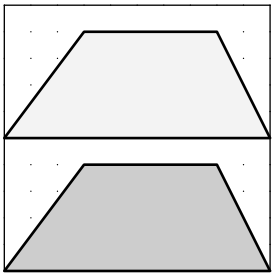
Линейные преобразования, допускаемые `mfpic`:

`\rotate`$\{\theta\}$	Вращает рисунок относительно начала координат на угол θ градусов
`\rotatearound`$\{p\}\{\theta\}$	Вращает рисунок относительно точки p на угол θ градусов
`\reflectabout`$\{p_1\}\{p_2\}$	Отражает рисунок относительно линии, соединяющей точки $p_1 \ldots p_2$
`\shift`$\{p\}$	Смещает начало координат вектором p
`\scale`$\{s\}$	Масштабирует обе координаты на множитель s
`\xscale`$\{s\}$	Масштабирует координату X на множитель s
`\yscale`$\{s\}$	Масштабирует координату Y на множитель s
`\zscale`$\{p\}$	Масштабирует обе координаты величиной p и поворачивает рисунок на угол p
`\xslant`$\{s\}$	Производит наклон вдоль оси X умножением Y
`\yslant`$\{s\}$	Производит наклон вдоль оси Y умножением X на s
`\xyswap`	Отражает рисунок относительно линии $Y = X$
`\boosts`$\{\chi\}$	Специальное увеличение относительно χ

Аффинному преобразованию подлежит фрагмент, заключенный внутри команд `\begincoords`...`\endcoords`. Если не имеется такого корпуса, тогда преобразования будут применяться глобально, и даже поперек среды `mfpic`, так что будьте внимательными, чтобы не забыть включить преобразование!

```
\begin{mfpic}[10]{0}{10}{0}{10}
\draw\shade\lclosed
```

```
\lines{(0,0),(3,4),(8,4),(10,0)}
\begin{coords}
\shift{(0,5)}\draw\shade[2pt]\lclosed
\lines{(0,0),(3,4),(8,4),(10,0)}
\end{coords}\end{mfpic}
```



5. Функции

Функциональные макрокоманды имеют два или более параметров. Они имеют не обязательный первый параметр, `<spec>`, который определяет является ли функция гладкая, или ломанная (как линии отрезков) — если `<spec>` равна `s`, то функция будет гладкой; если `<spec>` равна `p`, то функция будет ломанной; по умолчанию `<spec>` зависит от цели макрокоманды.

Один обязательный параметр содержит три значения `<min>`, `<max>` и `<step>`. Независимая переменная функции начинает с величины `<min>` шагом `<step>` пока не достигнет `<max>`

Существует один или более параметров `<expr>`, каждый из которых является выражением, описывающий функцию и написанный с единственной неизвестной независимой переменной. Выражение `<expr>` передается непосредственно в соответствующую макрокоманду METAPOST и там интерпретируются по METAPOST'им правилам для алгебраических выражений. Операции включают `+, -, *, /` и `**` (`x**y` $= x^y$), с (и) для группирования. Функции включают `round, floor, ceiling, abs, sqrt, sind, cosd, mlog` и `mexp`.

(*Замечание*: тригонометрические функции `sind` и `cosd` берут аргументы в градусах; `mlog(x)= 256 ln x`, и `mexp` является ее обратной.

Существуют другие операции и функции, но эти наиболее используемые при черчении.)

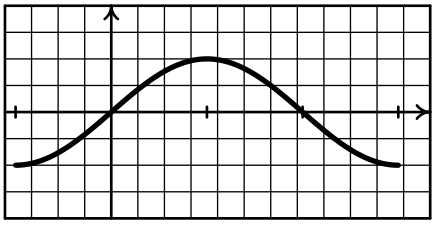
Вы можете также определить функцию $f(x)$ используя условное выражение {if <boolean>: <expr> else: < expr> fi }.

Отношения, имеющиеся для <boolean> части выражения, включают =, <, >, <= и >=.

```
\function[<spec>]{<xmin>,<xmax>,<step>}{<expr>}
```

Эта макрокоманда рисует зависимость $f(x)$=<expr>, где <expr> представляет собой выражение от переменной x. Значение по умолчанию для необязательного параметра <spec> равен s.

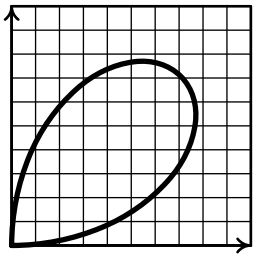
```
\begin{mfpic}[.4][20]{-100}{300}{-2}{2}
\axes\doaxes{lrbt}
\xmarks{-90,90,180,270}
\function{-90,270,4.5}{sind(x)}
\end{mfpic}
```



```
\parafcn[<spec>]{<tmin>,<tmax>,<step>}{(<expr0 >,<expr1 >)}
```

Эта макрокоманда рисует параметрическую зависимость (x(t); y(t)) =(<expr0 >,<expr1 >), где <expr0> и <expr1> представляют собой выражение от переменной t. Значение по умолчанию для необязательного параметра <spec> равен s.

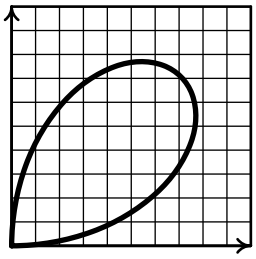
```
\begin{mfpic}[90]{0}{1}{0}{1}
\gridlines{.1,.1}
\pen{1pt}\doaxes{lrbt}
\axes\pen{2pt}
\parafcn{0,1,0.125}{sind(180t)*dir(90t)}
\end{mfpic}
```



```
\plrfcn[<spec>]{<tmin>,<tmax>,<step>}{(<expr>)}
```

Эта макрокоманда рисует параметрическую зависимость в полярных координатах: `r(t) =<expr>`, где `<expr>` представляет собой некоторое выражение от переменной `t`. Каждая величина `t` интерпретируется как угол, измеряемый в градусах. (Здесь «`t`» заменяют «theta»; т.к. многие из нас испытывают недостаток Греческих клавиатур). Значение по умолчанию для необязательного параметра `<spec>` равен `s`.

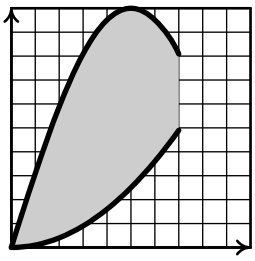
```
\begin{mfpic}[90]{0}{1}{0}{1}
\gridlines{.1,.1}
\pen{1pt}\doaxes{lrbt}
\axes\pen{2pt}
\plrfcn{0,90,10}{sind(2t)}
\end{mfpic}
```



```
\btwnfcn[<spec>]{<tmin>,<tmax>,<step>}{<expr0 >}{<expr1 >}
```

Эта макрокоманда рисует замкнутую фигуру, ограниченную двумя функциями $f(x)$=`<expr0>` и $g(x)$=`<expr1>`, где `<expr0>` и `<expr1>` представляют собой функции от переменной `x`. Область ограничивается также вертикальными линиями `<tmin>` и `<tmax>`. В отличие от вышеупомянутых функциональных макрокоманд, значение по умолчанию для `<spec>` равен `p` — эта макрокоманда предназначается для штриховки между функциями, задача, для которой гладкость не нужна.

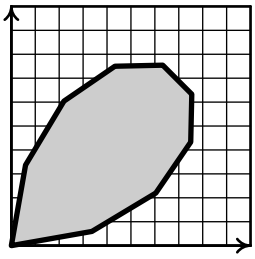
```
\begin{mfpic}[90]{0}{1}{0}{1}
\axes\shade
\btwnfcn{0,0.7,0.1}{x**2}{sind(180x)}
\function{0,0.7,0.1}{x**2}
\function{0,0.7,0.1}{sind(180x)}
\end{mfpic}
```



```
\plrregion[<spec>]{<tmin>,<tmax>,<step>}{<expr>}
```

Эта макрокоманда рисует замкнутую фигуру типа криволинейного сектора в полярных координатах, ограниченную кривой линией `r(t)=<expr>`, где `<expr>` представляет собой функцию от переменной `t`. Значение `t` — угол (измеряемый в градусах). Область также ограничивается прямыми лучами `<tmin>` и `<tmax>`, т.е. выходящими из центра. Значение по умолчанию для `<spec>` равен `p` — эта макрокоманда предназначается для штриховки области, задача, для которой гладкость не нужна.

```
\begin{mfpic}[90]{0}{1}{0}{1}
\gridlines{.1,.1}
\pen{1pt}\doaxes{lrbt}
\axes\pen{2pt}\draw\shade
\plrregion{0,90,10}{sind(2t)}
\end{mfpic}
```

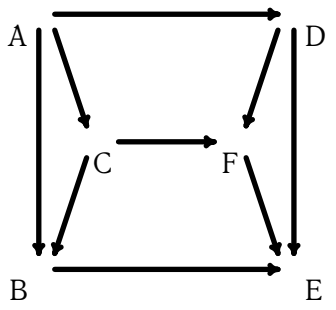


6. Метки и заголовки

Макрокоманды ...\tlabel, ...\tlabels, ...\axislabels и \tcaption не воздействуют на METAPOST файл (*.mp) вообще, но добавляются к рисунку LaTeX'ом. Поэтому, если эти макрокоманды изменить или добавить в документ, то не нужно повторять последние два шага (обработки с METAPOST или обработки с LaTeX) для того, чтобы закончить LaTeX документ.

```
\tlabel[<just>](<x>,<y>){<labeltext>}
\tlabels{<params1><params2>...}
```

Первая макрокоманда ставит LaTeX метку на графе. Без параметра [<just>], метка ставится с нижнего левого угла метки в точке (<x>,<y>). Специальная форма \tlabels применяется также как и \tlabel, только в аргументе содержит список параметров. То есть каждый параметр <params1> содержит набор параметров команды \tlabel. Необязательный параметр [<just>] определяет соответствующее смещение метки, что касается точки (<x>,<y>) — [<just>] это две характерные последовательности, где первая это одна из t (верхняя часть), c (центр), или b (нижняя часть), для того, чтобы определить вертикальное смещение, и вторая последовательность это одна из l (левая часть), c (центр), или r (правая часть), для определения горизонтального смещения. Ориентация по умолчанию равносильна определению [bl]. Если метка выходит за пределы графа в любом направлении,то бокс, содержащий граф, будет расширен для того, чтобы выделить участок памяти для метки.



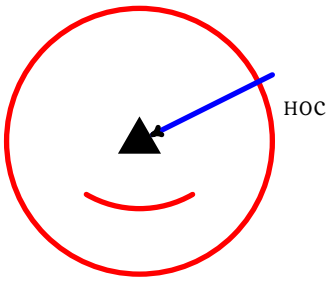
```
\begin{mfpic}[12][10]{.8}{10}{1}{9.2}
\tlabels{[br](1,9){A} [tr](1,1)
{B} [cc](3,5){C} [bl](9,9){D}
[tl](9,1){E}[cc](7,5){F}}
\end{mfpic}
```

```
\tcaption[<maxwd>,<linewd>]{<captiontext>}
```

Эта макрокоманда помещает заголовок в нижнюю часть рисунка. Необязательные параметры `<maxwd>` и `<linewd>` управляются автоматическим разбиением текста на стоки заданной длины. Параметр `<maxwd>` задает предельную относительную длину для строки, если строка превышает длину `<maxwd>` — размер графа, то она разбивается на несколько строк с длиной `<linewd>` — размер графа. По умолчанию для `<maxwd>` и `<linewd>` используются значения 1.2 и 1.0, соответственно. Если заголовок и граф имеют различные длин, то оба центрируются относительно друг друга. Если текст разбит на несколько строк, то каждая строка (за исключением последней) будет выравнена по левому и правому краю, но первая строка не будет делать отступ.

```
\begin{mfpic}[40]{-1}{1}{-1}{1}
\circle{(0,0),1}
\plotsymbol[10pt]{SolidTriangle}{(0,0)}%
{\pointfillfalse \point{(0.4,0.4),(-0.4,0.4)}}
\arc{(-0.4,-0.4),(0.4,-0.4),60}%
\arrow\lines{(1,0.5), (0.1,0.05)}%
\tlabel[Bl angle(.9,.45)](1,0.5){нос}
\tcaption{Смешная картинка }
\end{mfpic}
```

```
\everytlabel{<TEX-code>}
```



Смешная картинка

Устанавливает формат метки. Например, если вы хотите, чтобы все последующие метки были написаны жирным шрифтом, то необходимо написать `\everytlabel{\bf}`.

```
\tlabeljustify{<just>}
```

После этой команды все метки без параметра смещения будут определяться параметром `<just>`. Например, после использования команды `\tlabeljustify[cr]` команда `\tlabel` имеет смещение метки по умолчанию `[cr]`.

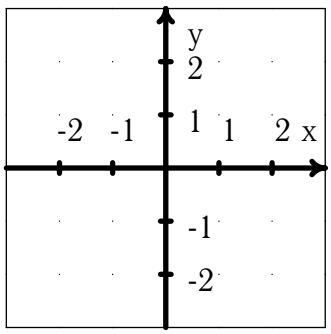
```
\tlabeloffset{<hlen>}{<vlen>}
```

Эта команда задает расстояние смещения метки для команды `\tlabel`. `<hlen>` — расстояние смещение метки вправо, `<vlen>` — расстояние смещения метки вверх (отрицательная длина будет соответственно смещение влево и вниз).

```
\axislabels{<axis>}[<just>]{{<txt1>}<n1>,%
                            {<txt2>}<n2>,. . .}
```

Эта команда располагает текст `<txtk>` в позиции `<nk>` на заданной оси `<axis>`, которая должна быть обозначена одной из букв `l, b, r, t, x` или `y`. Эта команда идентична `\tlabel`, за исключением того что значение по умолчанию параметра `<just>` зависит заданной оси (параметры настройки `\tlabeljustify` игнорируются). В случае граничных осей, значение по умочанию — это расположение метки с внешней стороны оси и по центру. Например, для нижней оси значение по умолчанию будет `[tc]`. Значение по умолчанию для осей `x` и `y` будет внизу и слева, соответственно. Параметр `<just>` может использоваться для изменения

смещения метки. Например, расположение метки `inside` левой граничной оси будет [`cl`].




```
\begin{mfpic}[10]{-6}{6}{-6}{6}
\everytlabel{\small}
\tlabeloffset{4pt}{6pt}
\axes \xmarks{-4,-2,2,4}
\axismarks{y}{-4,-2,2,4}
\axislabels{x}[bc]{{1}2,{2}4,
{-2}-4,{-1}-2,{x}5}
\axislabels{y}[tl]{{1}2,{2}4,
{-2}-4,{-1}-2,{y}5}
\end{mfpic}
```

```
\plottext[<just>]{<text>}{(x0,y0),(x1,y1);. . .}
```

Команда `\plottext` помещает текст `<text>` в каждой точке из списка (x_0, y_0), (x_1, y_1),.... Подобно `\axislabels`, эта команда игнорирует `\tlabeljustify`. Значение по умолчанию [`cc`], если параметр `<just>` отсутствует.

7. Параметры

Существует множество параметров для `mfpic`, которые пользователь может изменить, чтобы получить различные результаты (такие как различная форма и размер наконечника стрелки). Большинство параметров хранится LaTeX'ом как размерности. Таким образом они доступны глобально, даже если нет открытого METAPOST файла; их изменение есть объект обычных LaTeX правил. Некоторые параметры, однако, хранятся METAPOST'ом, таким образом их изменение не будет иметь результатов до тех

пор пока не будет открыт METAPOST файл, и изменение есть объект правил METAPOST'а. Для пользователя `mfpic`, это означает, что изменение внутри `\mfpic`. . . `\endmfpic` локально по отношению к этому окружению на другие LaTeX'ие группировки и не дает результата.

```
\mfpicunit
```

Эта величина сохраняет основную единичную длину для изображения `mfpic` — масштабы `x` и y в макрокоманде `\mfpic` являются кратными этой единичной длины. Значение по умолчанию — 1 пункт. Для изменения его значения необходимо применить команду

```
\pointsize
```

Эта величина определяет диаметр окружности, нарисованной макрокомандой `\point`. По умолчанию величина 2 пункта.

```
\pointfilled
```

Это логическая (булевская) переменная LaTeX'а, которая определяет, будут ли заполнятся черным цветом внутренняя область точек, рисуемых командой `\point` (если истина) или останется не заполненным (выведенный контур фигуры, уничтоженный фон). Значение по умолчанию является истина.

```
\pen[<pensize>]
```

Эта макрокоманда устанавливает толщину рисующего пера. По умолчанию величина 0.5 пунктов. Это количество определяет METAPOST.

```
\headlen
```

Эта величина определяет длину наконечника стрелки, нарисованного макрокомандой `\arrow`. По умолчанию величина 3 пункта.

```
\axisheadlen
```

Эта величина хранит длину наконечника стрелы, нарисованного макрокомандой `\axes`. По умолчанию величина 5 пунктов.

```
\headshape[<hdwdr>][<hdten>][<hfilled>]
```

Эта макрокоманда устанавливает форму наконечника стрелки, нарисованного макрокомандами \arrow и \axes. Величина <hdwdr> — это отношение ширины наконечника стрелки к ее длине; <hdten> — это натяжение кривых Bezier; и <hfilled> — это логическое значение, указывающее должны ли стрелки заполняться черным цветом (если истина) или остаться не заполненными. Значение по умолчанию является 1, 1, ложь, соответственно. Эта величина хранится METAPOST'ом. Установив <hdten> к «infinity» сделаем стороны наконечника прямыми линиями.

\dashlen, \dashspace

Эти размеры сохраняют, соответственно, длину пунктирной черточки и длину просвета между черточками для отрезков, которые рисуются макрокомандой \dotted. Просвет между черточками может быть подправлен макрокомандой \dotted не более чем на $\frac{dashspace}{n}$, где n — число просветов на отрезке, для того, чтобы на концах не получились кусочки штриховки. По умолчанию величины — 4 пункта.

\dashlineset, \dotlineset

Эти макрокоманды предусматривают подходящий образец для величин \dashlen и \dashspace. Макрокоманда \dashlineset устанавливает оба значения в 4 пункта; макрокоманда \dotlineset устанавливает \dashlen в 1 пункт и \dashspace в 2 пункта.

\hashlen

Эта величина хранит длину оси с метками, нарисованная макрокомандами \marks и \ymarks. По умолчанию величина 4 пункта.

\shadespase

Эта величина устанавливает расстояние между точками, нарисованная макрокомандой \shade. По умолчанию величина 1 пункт.

\darkershade, \lightershade

Обе эти макрокоманды изменяют величину \shadespase с постоянными коэффициентами, $\frac{5}{6}$ и $\frac{6}{5}$ соответственно, для того, чтобы обеспечить удобные стандартные установки для нескольких видов штриховки.

```
\hatchspace
```

Эта величина устанавливает интервал между строками, выведенный макрокомандой \hatch. Значение по умолчанию — 3 пункта.

8. Цветная графика в MFPIC

Чтобы работать с цветом, в преамбуле входного файла необходимо загрузить дополнительный пакет:

```
\usepackage[dvips]{color}
```

Все макрокоманды, описанные выше работают и в этом случае, необходимо только дополнительно определить цвет рисунков. Существует шесть команд, определяющих цвет: \drawcolor, \fillcolor, \hatchcolor, \headcolor, \tlabelcolor и \background. Они управляют цветами, используемых для рисования линий и кривых, для заполняющихся областей, для штриховки, для стрелок, и для текста.

```
\drawcolor[<color>]
\fillcolor[<color>]
\hatchcolor[<color>]
\headcolor[<color>]
\tlabelcolor[<color>]
\backgroundcolor[<color>]
```

Эти команды устанавливает цвет для соответствующих фигур. Макро \gfill использует \fillcolor. Все команды штриховки внутренней области фигуры, \xhatch, \rhatch, и т.д., используют \hatchcolor. Все рисунки кривых, линий, и т.д., включая их пунктирное изображение, а такжы оси координат и пунктирные границы фигур используют \drawcolor. Все стрелки, а также стрелки, образованные командой \axes, т.е. на осях координат, имеют цвет, определенный в \headcolor. Все текст, помещенный командой \tlabel используют цвет, определенный в \tlabelcolor.

Команда `\gclear`, т.е. заполнение цветом замкнутой фигуры, использует `\backgroundcolor`.

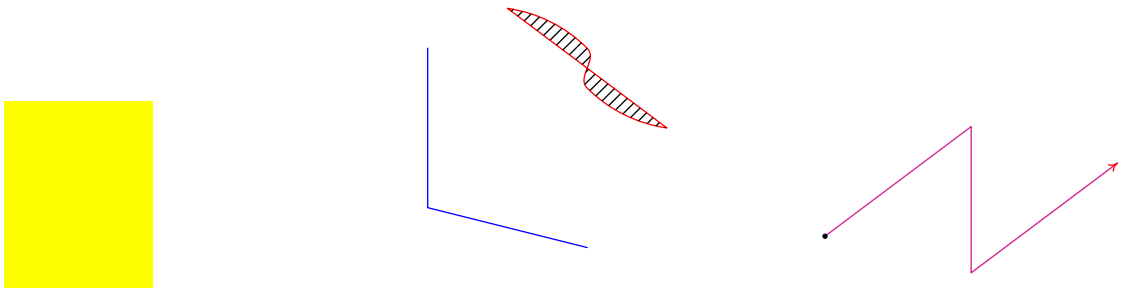
Ниже несколько примеров, как определяется цвет для отдельных элементов рисунка:

```
\begin{mfpic}[40]{0}{2}{0}{1}
\mfpdefinecolor{DarkPuce}{rgb}{.8,.12,.56}
\fillcolor{green}\drawcolor{DarkPuce}
\headcolor{red}
\arrow \lines{(0,0.35),(1,1.1),
1,0.1),(2,0.85)} \point{(0,0.35)}
\end{mfpic}
```

Более того, для управление над цветом индивидуальных элементов рисунка к макрокомандам `\gfill` и `\draw` была добавлена дополнительная опция определения цвета.

```
\gfill[<color>]
\draw[<color>]
```

```
\begin{mfpic}[15]{0}{1}{0}{1}
\gfill[yellow] \rect{(0,0),(0.8,1)}
\end{mfpic}
```



```
\begin{mfpic}[10]{-1}{1}{-1}{1}%
\rhatch\draw[red]\lclosed%
\curve{(1,0),(0.5,0.25),(0.5,0.5),(0,0.75)}%
\lines{(-0.5,0.5),(-0.5,-0.5),(0.5,-0.75)}
\end{mfpic}
```

Лекция 25 ⏮

Язык METAPOST'а

Типы данных, основные операторы и графические примитивы для рисования в языке METAPOST.

1. Типы данных

METAPOST имеет десять типов данных: числовой, для пар, для путей, трансформации, цвета (rgb-цвета), cmyk-цвета, строковый, логический, для картинок и тип пера.

Тип *pair* (пары) представляется как пара числовых количеств. Пары используются для задания координат в команде *draw*. Пары можно складывать, вычитать, использовать в выражениях усреднения, умножать и делить на числа.

Тип *path* (путей) — важные отдельные объекты, которые можно сохранять и изменять. Путь представляет прямую линию или кривую, определяемые параметрически.

Другой тип данных представляет произвольную *аффинную трансформацию* (transform). Трансформация может быть любой комбинацией вращений, масштабирований, наклонов и сдвигов.

Если `p = (px; py)` — это пара и `T` — это трансформация, то `p transformed T` — это пара вида

(tx + txxpx + txypy; ty + tyxpx + tyypy);

где 6 числовых количеств `(tx; ty; txx; txy; tyx; tyy)` определяют `T`.

Трансформации могут быть применимы к путям, рисункам, перьям и трансформациям.

Тип *цвета* (color) подобен типу пары, но он имеет три компоненты вместо двух и каждая компонента обычно находится в диапазоне от 0 до 1. Подобно парам, цвета могут складываться, вычитаться, использоваться в выражения усреднения, умножаться и делиться на числа. Цвета могут

задаваться при помощи констант `black, white, red, green, blue` или явно заданными красной, зеленой и синей компонентами.

Черный — это `(0,0,0)` и белый — это `(1,1,1)`. Уровень серого, такой как `(.4,.4,.4)`, можно также задать как `0.4white`. Хотя цветовой переменной может быть любая упорядоченная тройка, при добавлении объекта к картинке METAPOST преобразует ее цвета обрезкой каждой цветовой компоненты до диапазона от 0 до 1. Например, METAPOST будет выводить цвет `(1,2,3)` как `(1,1,1)`.

METAPOST решает линейные уравнения с цветами таким же образом как и с парами. Тип `'rgbcolor'` (rgb-цвет) — это синоним типа `'color'` (цвет).

Тип *cmykcolor* (cmyk-цвета) подобен типу `color`, но имеет четыре компоненты вместо трех. Этот тип используется для задания цветов их зеленоголубой (cyan), пурпурнокрасной (magenta), желтой и черной компонентами. Из-за того, что cmyk-цвет использует краски вместо световых лучей, белый цвет будет выражаться как `(0,0,0,0)` и черный как `(0,0,0,1)`.

Теоретически цвета `(c,m,y,1)` и `(1,1,1,k)` должны давать черный для любых значений c, m, y и k. На практике этого избегают, т. к. это тратит цветные чернила и может приводить к неудовлетворительным результатам.

Строки (string) представляют последовательность символов. Строковые константы задают- ся в двойных кавычках "подобно этой". Строковые константы не могут содержать двойных кавычек или переходов на новую строку, но есть способ конструировать строки, содержащие любую последовательность восьмибитных знаков.

Логический тип (boolean) имеет константы `true` и `false` и операторы `and, or, not`. Отношения `=` и `<>` проверяют объекты любых типов на равенство и неравенство. Отношения сравнения `<, <=, >` и `>=` определяются словарно для строк и обычным способом для чисел.

Тип данных *picture* (рисунок) — это в точности то, что подразумевается его именем. Все, что можно нарисовать в METAPOST, можно сохранить в

переменной-картинке. Фактически команда `draw` сохраняет свои результаты в особой переменной-картинке, называемой currentpicture. Картинки могут быть добавлены к другим картинкам и их можно трансформировать.

Наконец, тип данных, называемый `pen` (перо). Главная функция перьев в METAPOST в определении толщины линии, но их также можно использовать для достижения каллиграфических эффектов.

Обычно выражение-перо имеет форму `pencircle scaled` [числовое значение]. Оно определяет круговое перо, производящее линии постоянной толщины.

2. Операторы

Операторы умножения и деления, * и /, — примеры того, что называется *первичным бинарным оператором*. Каждый из них может допускать *два числовых операнда* или один числовой *операнд* и один типа пара или цвет.

Оператор возведения в степень ** — это *первичный бинарный оператор*, который требует два числовых *операнда*. Размещение его на том же уровне приоритета, что и умножение и деление, имеет неприятное последствие, состоящее в том, что $3 * a * *2$ значит $(3a)^2$, а не $3(a^2)$.

Из-за того, что унарный минус относится к первичному уровню, он также приводит к неудобочитаемости типа $-a * *2$, означающей $(-a)^2$ К счастью, вычитание имеет меньший приоритет и $a - b * *2$ означает $a - b^2$, вместо $(a - b)^2$.

Другим первичным бинарным оператором является оператор `dotprod`, вычисляющий скалярное произведение двух пар. Например, `z1 dotprod z2` эквивалентно

```
x1*x2 + y1*y2.
```

Аддитивные операторы - и + — *вторичные бинарные операторы*, применимые к числам, парам или цветам и производящие результаты того же типа.

Другие операторы, что попадают в эту категорию — это «Пифагорово сложение» $++$ и «Пифагорово вычитание» $+-+$: $a++b$ значит $\sqrt{a^2+b^2}$ и $a+-+b$ значит $\sqrt{a^2-b^2}$.

Есть еще много других операторов для перечисления, но одни из самых важных — это логические операторы `and` и `or`. Оператор `and` — это первичный бинарный оператор и оператор `or` — это вторичный бинарный оператор.

METAPOST имеет некоторые встроенные полезные `numeric` функции. Они включают: `length, sqrt, sind, cosd, mlog, mexp, floor, angle, uniformdeviate, normaldeviate.`

`length` x — абсолютное значение x;

`sqrt` x — квадратный корень x;

`sind` x — синус градуса x;

`cosd` x — косинус градуса x;

`angle` (x, y) — арктангенс y/x;

`mlog` x равный $256 \ln x$, где $\ln$ — логарифм по основанию e;

`mexp` x равный $e^{x/256}$.

`floor` x — самое большое целое число меньше или равное x.

METAPOST понимает `ceiling`, `abs`(то же самое, что и `length`), `round`, `min` и `max`.

3. Переменные

METAPOST позволяет составные имена переменных, такие как `z.a, x2r, y2r` и `z2r`, где `z2r` означает `(x2r,y2r)`, а `z.a` — `(x.a,y.a)`. Фактически существует широкий класс суффиксов, например, `z<суффикс>`, означающий `(x<суффикс>; y<суффикс>)`.

Декларации переменных.

Имя переменной — это либо символьный знак, либо последовательность символьных знаков. Большинство символьных знаков являются правильными именами переменных, но что угодно с предопределенным значением подобно `draw, +` или `..` недопустимо, например, имена пере-

менных не могут быть макросами или примитивами METAPOST. Это второстепенное ограничение допускает широкий класс имен переменных: `alpha, ==>, @&#$&` и `~~` — все они легитимные имена переменных.

Переменные, подобные `x2` и `y2`, обычно имеют числовое значение, поэтому мы можем использовать факт, что `z<суффикс>` — это сокращение для

```
(x<суффикс>; y<суффикс>);
```

для генерации пар-значений, когда нужно.

Объявления типа делает возможным использование почти любой схемы именования при удалении всех предшествующих значений, что могут вызвать взаимодействие. Например, декларация `pair pp, a.b;` делает `pp` и `a.b` неизвестными парами. Такая декларация не является строго локальной, т. к. `pp` и `a.b` не восстанавливают автоматически свои предшествующие значения в конце текущего рисунка. Они опять становятся неизвестными парами при повторении этой декларации.

Декларации работают одинаковым образом для любого другого из оставшихся девяти типов: числового, путевого, трансформационного, цветового, cmyk-цветового, строкового, логического, рисуночного и перьевого. Единственное ограничение в том, что вы не можете задать точный числовой индекс в декларации переменной. Не нужно писать ошибочных деклараций типа `numeric q1, q2, q3;` используйте обобщенный символ индекса [ ] вместо чисел для объявления всего массива: `numeric q[ ]`. Можно также определить “многоразмерные” массивы. После декларации

```
path p[ ]q[ ], pq[ ][ ];
```

`p2q3` и `pq1.4 5` — это два пути.

4. Линейные уравнения

Важным свойством METAPOST’а является возможность решать линейные уравнения, вследствие этого программы могут писаться в частично декларативной манере. Например, METAPOST может считать

$$a + b = 3; \quad 2a = b + 3;$$

и вывести, что $a = 2$ и $b = 1$. Эти же выражения могут быть записаны слегка более компактно путем соединения их вместе несколькими знаками равенства:

$$a + b = 2a - b = 3;$$

Каким бы способом вы не задавали уравнения, вы можете затем дать команду

```
show a,b;
```

для просмотра значений a и b. METAPOSTответит, напечатав

» 2

» 1

Заметьте, что = не является оператором присваивания; он просто объявляет, что левая часть равна правой. Таким образом, $a = a + 1$ приведет к сообщению об ошибке, жалующемуся на “противоречивое уравнение”. Способ увеличения значения `a` — в использовании оператора присваивания `:=` как в примере:

```
a:=a+1;
```

Другими словами, `:=` служит для изменения существующих значений, `a =` служит для задания линейных уравнений для решения.

Нет ограничений на смешивание уравнений и операций присваивания, например,

```
a = 2; b = a; a := 3; c = a;
```

После первых двух уравнений, устанавливающих `a` и `b` равными 2, операция присваивания изменит `a` на 3 без влияния на `b`. Окончательное значение `c` — 3, т. к. оно приравняется новому значению `a`.

В общем, операция присваивания интерпретируется сначала вычислением нового значения и затем уничтожением старого значения из всех существующих уравнений перед собственно присваиванием.

5. Уравнения и координатные пары

METAPOST может также решать линейные уравнения, содержащие координатные пары. Примером этого может быть уравнение, подобное

```
z1=(0,.2in).
```

Каждая сторона уравнения должна быть сформирована сложением или вычитанием координатных пар и умножением или делением их на известные числовые количества. Рассмотрим именование вида

```
z<число>,
```

которое весьма удобно, потому что оно — сокращение для

```
(x<число>, y,число>).
```

Это делает возможным давать значения переменным `z` заданием уравнений для их координат. Например, точки `z1, z2, z3,` и `z6` можно инициализировать следующими уравнениями:

```
z1=-z2=(.2in,0);  x3=-x6=.3in;  x3+y3=x6+y6=1.
```

В точности те же самые точки могли быть получены прямой установкой их значений:

```
z1=(.2in,0); z2=(-.2in,0);
z3=(.3in,.8in); z6=(-.3in,1.4in);
```

После чтения уравнений METAPOST знает значения `z1, z2, z3` и `z6`.

Следующий шаг в конструировании точек — усредняющая конструкция. Она имеет вид `z4=1/3[z3,z6],` который означает, что то точка z4 прошла 1/3 пути от z3 до z6, т. е. `z4 = z3 +1/3(z6-z3).`

Усреднение может быть также использовано для того, чтобы сказать, что некоторая точка находится в неизвестной позиции на прямой между двумя известными точками. Например, мы можем ввести новую переменную aa и записать что-то вроде

z20=aa[z1,z3];

Это означает, что z20 — это неизвестное отношение aa пути по прямой между z1 и z3. Еще одного такого отношения для другой линии достаточно для фиксации значения z20.

Описание того, что z20 пересечение прямых z1-z3 и z2-z4 требует введения еще одной переменной ab и устанавливается равенством z20=ab[z2,z4];

Это позволяет METAPOST'у найти x20, y20, aa и ab.

Поскольку сложно постоянно думать о новых именах, подобных aa и ab, можно использовать специальную возможность, называемую `whatever`. Этот макрос генерирует новую анонимную переменную каждый раз, когда он появляется. Таким образом, команда z20=whatever[z1,z3]=whatever[z2,z4] устанавливает z20 как и раньше, но она использует `whatever` для генерации двух различных анонимных переменных вместо aa и ab.

Уравнения в системе могут быть заданы в любом порядке, но все уравнения должны быть линейными и все переменные должно быть возможным вычислить тогда, когда они понадобятся. Это значит, что уравнения

z1=-z2=(.2in,0); x3=-x6=.3in; x3+y3=x6+y6=1.1in;

z4=1/3[z3,z6]; z5=2/3[z3,z6];

достаточны для определения z1 и остальных до z6 и порядок уравнений не важен.

Основные команды для рисования приведены в следующей таблице:

Имя	Объяснение
`draw`	Рисовать линию или картинку
`drawarrow`	Рисовать линию со стрелкой на конце
`drawdblarrow`	Рисовать линию со стрелками в обоих концах
`fill`	Заполнить циклический путь
`filldraw`	Рисовать циклический путь и заполнить его внутри
`undraw`	Стереть линию или рисунок
`unfill`	Стереть внутри замкнутого пути
`unfilldraw`	Стереть циклический путь и все внутри него
`withcolor`	Используй обычный цвет в команде рисования
`withpen`	Используй перо в команде рисования

6. Примеры

В приведенных ниже примерах мы рисуем картинки, используя пакет `mfpic`, о котором шла речь в предыдущей лекции, все вычисления передаем на работу в METAPOST, погружая необходимые данные в виде

аргумента команды `\mfsrc`. Сам пакет `mfpic` эти вычисления проделать не может!

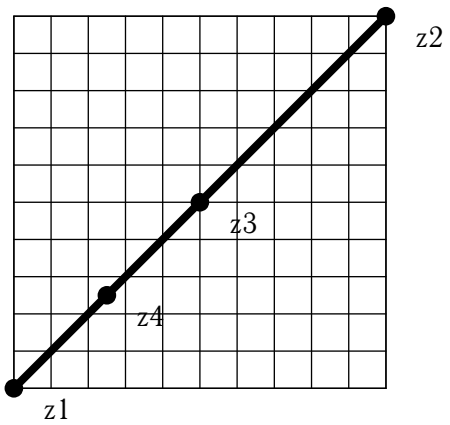
6.1. Определение точки

```
z1=(x1,y1);\ z2=(x2,y2);
```

Этой командой мы определяем точки $z1$ и $z2$, где $(x1, y1)$ и $(x2, y2)$ - координаты точек $z1$ и $z2$ соответственно. Заметьте, что после каждого командной строки необходимо ставить ; (точку с запятой). В дальнейшем уже не будет необходимости писать координаты этих точек.

METAPOST легко выполняет задачу нахождения точки середины отрезка, 1/4 отрезка и так далее...

```
\begin{mfpic}[10]{0}{10}{0}{10}
\doaxes{lrtb}
\mfsrc{z1=(0,0); z2=(10,10);
z3=1/2[z1,z2]; z4=1/4[z1,z2];}
\point[2pt]{z1,z2}
\end{mfpic}
```



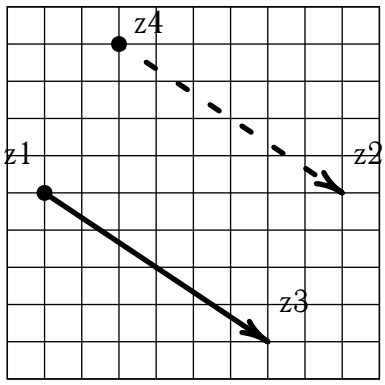
6.2. Нахождение точки из заданных условий

METAPOST может находить точки из уже определенных точек. Например, если у нас есть вектор $(z1, z2)$, и на необходимо построить такой же вектор из точки $z3$, то мы должны написать:

```
$z1$=$(x1,y1)$;  $z2$=$(x2,y2)$;
$z3$=$(x3,y3)$;  $z1-z2$=$z3-z4$
```

где точки $z1$, $z2$ и $z3$ уже заданы и вектор $(z3, z4)$ должен быть равен вектору $(z1, z2)$, то есть точка $z4$ вычисляется самостоятельно.

```
\begin{mfpic}[12]{0}{10}{0}{10}
\mfsrc{z1=(1,5); z2=(9,5);
z3=(7,1); z3-z1=z2-z4}%
\arrow[l10pt]\lines{z1,z3}
\arrow[l10pt]\dashed[4pt,10pt]
\lines{z4,z2} \point{z1}
\fillcolor{Red}\point{z4}
\end{mfpic}
```

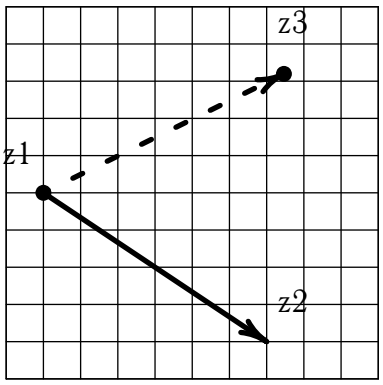


Если с помощью двух заданных точек $z1$ и $z2$ мы хотим найти третью $z3$, отстоящую от точки $z1$ на таком же расстоянии, что и точка $z2$, но в другом заданном направлении, мы должны использовать:

```
z1=(x1,y1);z2=(x2,y2);z1-z2=(z3-z2) rotated \theta
```

где точка $z3$ вычисляется самостоятельно. Она находится на расстоянии $(z1, z2)$ от точки $z1$, под углом θ по часовой стрелке от направления вектора $(z1, z2)$.

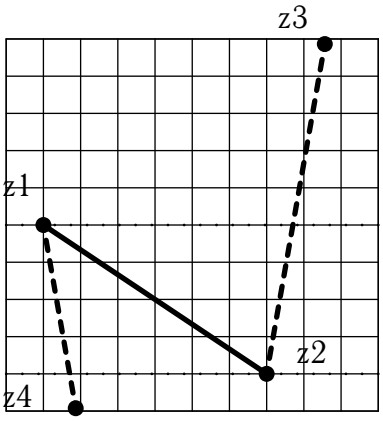
```
\begin{mfpic}[12]{0}{10}{0}{10}
\mfsrc{z1=(1,5); z2=(7,1);
z2-z1=(z3-z1) rotated 300}
\arrow[l10pt]\lines{z1,z2}
\arrow[l10pt]\dashed[4pt,10pt]
\lines{z1,z3}\point{z1}\fillcolor{Red}
\point{z3}  \end{mfpic}
```



$z1{=}(x1, y1)$; $z2{=}z1{+}\xi$ dirθ или $z1{=}(x_1, y_1)$; $z2{=}z1{-}\xi$ dirθ

Эти команды находят точку $z2$, которая находится на расстоянии ξ (измеряется в единицах системы координат) от точки $z1$, и угол между вектором $(z1, z2)$ и горизонтальной линией должен быть равен θ.

```
\begin{mfpic}[12]{0}{10}{0}{10}
\mfsrc{z1=(1,5);z2=(7,1);z3=z2+9dir80;z4=z1-5dir100;}
\lines{z1,z2}\point{z1,z2}\fillcolor{Red}\point{z3,z4}
\dashed[4pt,4pt]\lines{z2,z3}
\dashed[4pt,4pt]\lines{z4,z1}          \end{mfpic}
```



$z1 = (x1, y1)$; $z2 = (x_2, y_2)$; $z3 - z1{=}\mathit{whatever}{*}\mathit{dir}\theta$; $z3 - z2{=}\mathit{something}{*}\mathit{dir}\xi$;

Этими командами мы находим точку из уравнений

$z3 - z1{=}\mathit{whatever}{*}\mathit{dir}\theta$ и $z3 - z2{=}\mathit{something}{*}\mathit{dir}\xi$.

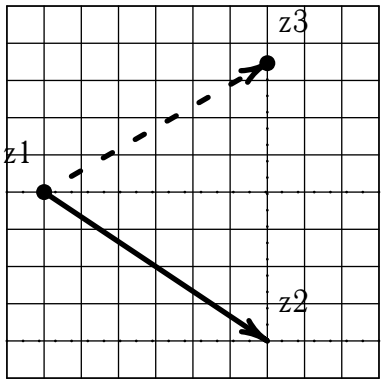
Первое уравнение означает, что угол между вектором $(z3, z1)$ и горизонтальной линией должен быть равен θ градусов. Угол отсчитывается от горизонтальной линии против часовой стрелки. Второе уравнение задает угол между вектором $(z3, z2)$ и горизонтальной линией, который должен

быть равен ξ. Причем здесь угол может определяться как по часовой стрелки относительно горизонтальной линии, так и против часовой стрелки. Это определяется тем, как задано первое уравнение, т.е. команда *something* зависит от *whatever*, так чтобы лучи пересекались.

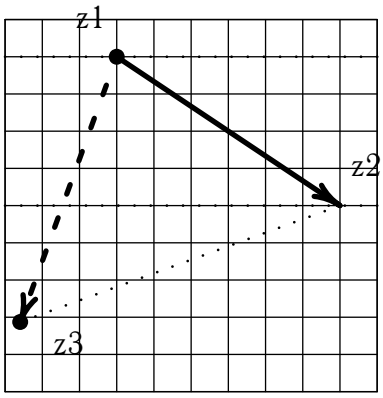
Эти уравнения равносильны уравнениям

$z3 - z1 = whatever * dir\theta$ и $z3 - z2 = whatever * dir\xi$,

если мы точно знаем, что заданные лучи будут пересекаться. Таким образом мы задаем два направления, на пересечении которых и будет находиться точка $z3$.



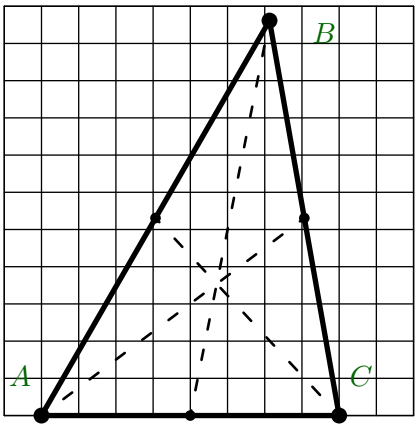

```
\begin{mfpic}[12]{0}{10}{0}{10}
\mfsrc{z1=(1,5);z2=(7,1);z3-z2=whatever*dir 90;
z3-z1=whatever*dir 30;}%
\arrow[l10pt]\lines{z1,z2}
\arrow[l10pt]\dashed[4pt,10pt]\lines{z1,z3}%
\point{z1} \fillcolor{Red}\point{z3}%
\dotted[1pt,6pt]\lines{z2,z3}
\end{mfpic}
```



```
\begin{mfpic}[12]{0}{10}{0}{10}
\mfsrc{z1=(3,9);z2=(9,5);z3-z2=whatever*dir20;
z3-z1=something*dir70;}
\arrow[l10pt]\lines{z1,z2}\arrow[l10pt]
\dashed[4pt,10pt]\lines{z1,z3} \point{z1}
\fillcolor{Red}\point{z3}
\dotted[1pt,6pt]\lines{z2,z3}
\end{mfpic}
```

6.3. Построение медианы треугольника

Для того чтобы построить медианы необходимо найти середины сторон заданного треугольника и провести прямые к этим точкам из вершин треугольника.



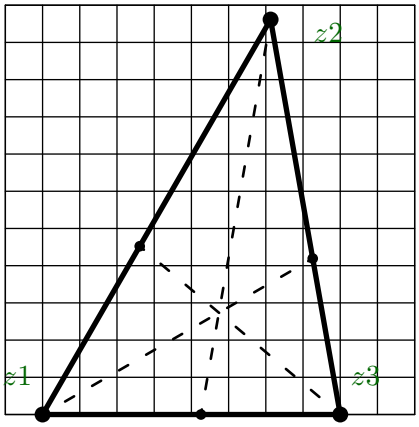

```
\begin{mfpic}[12]{0}{11}{0}{11}
\mfsrc{z1=(1,0); z3=(9,0);
z2-z3=whatever*dir 100;
z2-z1=whatever*dir 60;}%
\lines{z1,z2,z3,z1} \point{z1,z2,z3}
\point{(0.5[z1,z2]),(0.5[z1,z3]),(0.5[z2,z3])}
\dashed\lines{z3,(0.5[z1,z2])}
\dashed\lines{z2,(0.5[z1,z3])}%
\dashed\lines{z1,(0.5[z2,z3])}%
\end{mfpic}%A=z1, B=z2, C=z3
```

6.4. Построение биссектрисы треугольника

Чтобы построить биссектрису угла треугольника необходимо применять уравнения для нахождения соответствующей точки на противоположной стороне.

$$z4 = z1 + y * (dir(0.5(angle(z2 - z1) + angle(z3 - z1)))) = x[z2, z3];$$

где x и y - неизвестные, $z1$, $z2$ и $z3$ - заданные вершины треугольника, $angle(z2 - z1) + angle(z3 - z1)$ - угол между сторонами треугольника $z1z2$ и $z3z1$. $(z4, z1)$ - полученная биссектриса из вершины $z1$ заданного треугольника.



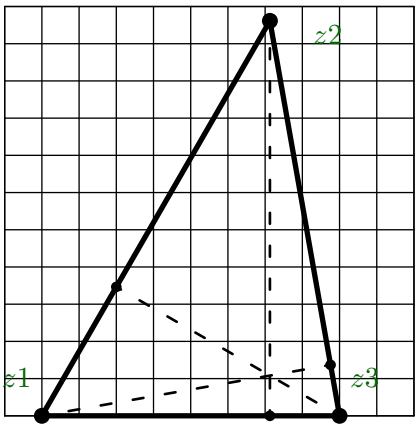

```
\begin{mfpic}[12]{0}{11}{0}{11}
\mfsrc{z4=z1+y*(dir(0.5(angle(z2-z1)+
 angle(z3-z1))))=x[z3,z2];}
\mfsrc{z5=z3+yy*(dir(0.5(angle(z1-z3)+
 angle(z2-z3))))=xx[z2,z1];}
\mfsrc{z6=z2+yyy*(dir(0.5(angle(z1-z2)+
angle(z3-z2))))=xxx[z3,z1];}
\end{mfpic}
```

6.5. Построение высоты треугольника

Для построения высоты определим следующую функцию

def $Ort(expr\ za, zb, zc) = numeric\ xa, yb;\ pair\ Base;$
$Base = xa[zb, zc] = za + yb * ((zb - zc)\ rotated 90);$ enddef

где $Base$ - это искомая точка, za, zb и zc - вершины треугольника.




```
\begin{mfpic}[12]{0}{11}{0}{11}
\gridlines{1,1} \doaxes{lrbt}
\pointsize=6pt  \pen{2pt}
\dashlen=4pt    \dashspace=8pt
\mfsrc{z1=(1,0);z3=(9,0);
z2-z3=whatever*dir 100;
z2-z1=whatever*dir 60;}%
\lines{z1,z2,z3,z1}   \point{z1,z2,z3}
\fillcolor{Red}
\pen{1pt}             \pointsize=4pt
\mfsrc{def Ort (expr za, zb,zc)=
numeric xa,yb;pair Base;
Base=xa[zb,zc]=za+yb*((zb-zc)rotated90);
enddef;}%
\mfsrc{ Ort (z1,z2,z3);}%
\dashed\lines{z1,Base}\point{Base}
\mfsrc{ Ort (z2,z3,z1);}
\dashed\lines{z2,Base}\point{Base}
\mfsrc{Ort (z3,z2,z1);}
\dashed\lines{z3,Base}\point{Base}
\tlabels{[br] (1,0.5) {$z1$}
         [bl] (9,0.5) {$z3$}(8,10){$z2$}}
\end{mfpic}
```

Лекция 26

Кривые в METAPOST'е

Параметризация кривых, основные команды и приемы для построения кривых.

1. Основные команды и приемы

Пути в METAPOST'е определяются как кусочные кубические кривые $(X(t), Y(t))$, где диапазон t от нуля до числа отрезков кривой. Другими словами, путь задается, как проходящий через последовательность точек, где $t = 0$ в первой точке, $t = 1$ в следующей, $t = 2$ в следующей и т. д.

Имеется несколько команд или операторов, которые позволяют извлекать информацию о кривых и применять ее для построений. Рассмотрим некоторые из них.

Оператор `intersectiontimes` — один из нескольких операторов, работающих с путями параметрически. Результатом

```
a intersectiontimes b
```

будет $(-1, -1)$, если пересечения нет; в противном случае, вы получите пару (t_a, t_b), где t_a — это время на пути `a`, когда он пересекает путь `b`, и t_b — это соответствующее время на пути `b`.

Например, предположим, что путь `a` обозначен тонкой линией на рис. 26.1 и путь `b` обозначен более толстой линией. Если метки показывают значения времени на путях, то пара значений времени, вычисленная в «`a intersectiontimes b`» должна быть одной из

$$(0.25, 1.77),\ (0.75, 1.40)\ (2.58, 0.24)$$

и какая из трех будет выбрана зависит от интерпретатора `MetaPost`. Точное правило выбора из многих точек пересечения достаточно сложное, но в данном примере вы получите $(0.25, 1.77)$. Меньшие значения времени предпочтительнее больших, так что (t_a, t_b) предпочтительнее, чем (t'_a, t'_b) пока $t_a < t'_a$ и $t_b < t'_b$. Когда нет простого способа минимизировать обе

компоненты t_a и t_b, t_a получает больший приоритет, но правила становятся более сложными, когда между t_a и t'_a нет целых чисел. (Больше деталей см. в *The METAFONTbook*.)

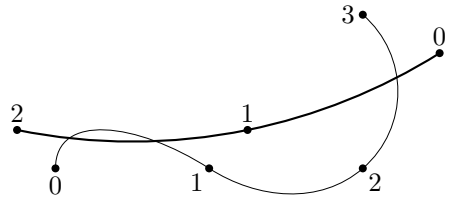


Рис. 26.1.
Два пересекающихся пути с отметками значений времени на каждом

Оператор `intersectiontimes` очень гибкий, потому что существует много всего, что можно сделать со значениями времени на пути.

Оператор `intersectionpoint` использует результат вычисления оператора `intersectiontimes`. Результатом

```
a intersectionpoint b
```

будет точка пересечения пути `a` пути `b`, соответствующая значениям времени, найденным оператором `intersectiontimes`.

Одно из самых важных — это задать вопрос: “Где проходит путь `p` во время `t`?” Оператор

```
point t of p
```

отвечает на этот вопрос. Если `t` меньше нуля или больше значения времени, присвоенного последней точке на пути, то конструкция `point of` обычно возвращает последнюю точку пути. Поэтому принято использовать предопределенную константу `infinity` (равную 4095.99998) как числовое выражение t в конструкции `point of`, когда имеем дело с концом пути.

Такое «бесконечное» значение времени не работает для циклического пути, т. к. значения времени, выходящие за нормальный диапазон, могут в этом случае обрабатываться модульной арифметикой, т. е. для циклического пути `p` через точки z_0, z_1, z_2, ..., z_{n-1} с обычным диапазоном параметров $0 \leqslant t < n$,

```
point t of p
```

может быть вычислено для любого t предварительным взятием t по модулю n.

Если модуль n недоступен, то оператор

```
length  p
```

дает целое значение верхнего предела нормального диапазона параметра-времени для заданного пути.

MetaPost использует такие же соответствия между значениями времени и точками на пути при вычислении оператора `subpath`. Синтаксис этого оператора

```
subpath (t1,t2)  of p
```

Результат будет подпуть, который лежит на пути p от `point` t_1 `of` p до `point` t_2 `of` p. Если $t_2 < t_1$, то подпуть идет обратно вдоль p.

Важный оператор, основанный на операторе `subpath`, — это оператор `cutbefore`. Для пересекающихся путей p_1 и p_2,

```
p1    cutbefore  p2
```

эквивалентно

```
subpath (xpart(p1 intersectiontimes p2),length p1) of p1
```

с тем исключением, что он также устанавливает переменную-путь `cuttings` в часть p_1, что отбрасывается. Другими словами, `cutbefore` возвращает свой первый аргумент без части до пересечения. При множественных пересечениях он пытается отбросить наименьшую часть. Если пути не пересекаются, то `cutbefore` возвращает свой первый аргумент.

Есть также аналогичный оператор, называемый `cutafter`, который работает с применением `cutbefore` с инвертированным временем вдоль

своего первого аргумента. Таким образом, p_1 `cutafter` p_2 пытается отрезать часть p_1 после последнего пересечения с p_2.

Другой оператор

```
direction  t  of p
```

находит вектор в направлении касательной к пути в точке, соответствующей времени t. Он определяется для любого времени, подобно конструкции `point of`. Вектор-результат имеет правильное направление и несколько произвольной размер. Соединение конструкций `point of` и `direction of` дает уравнение для линии тангенса, как показано на рис. 26.2.

```
beginfig(26);
numeric scf, #, t[];
3.2scf = 2.4in;
path fun; # = .1;
fun = ((0,-1#)..(1,.5#){right}..(1.9,.2#)%
      {right}..{curl .1}(3.2,2#))
  yscaled(1/#) scaled scf;
x1 = 2.5scf;
for i=1 upto 2:
  (t[i],whatever) =
fun intersectiontimes
((x[i],-infinity)--(x[i],infinity));
z[i] = point t[i] of fun;
z[i]-(x[i+1],0) = whatever*direction t[i] of fun;
draw (x[i],0)--z[i]--(x[i+1],0);
fill fullcircle scaled 3bp shifted z[i];
endfor
label.bot(btex $x_1$ etex, (x1,0));
label.bot(btex $x_2$ etex, (x2,0));
label.bot(btex $x_3$ etex, (x3,0));
draw (0,0)--(3.2scf,0);
pickup pencircle scaled 1pt; draw fun;
endfig;
```

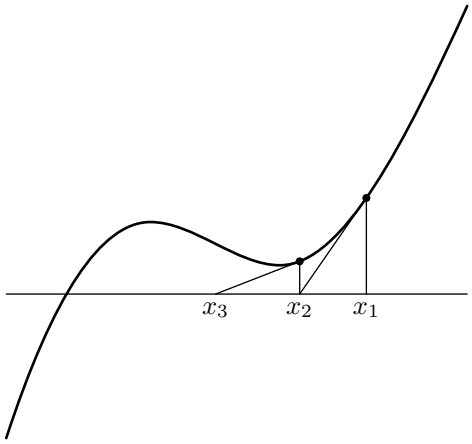


Рис. 26.2.
Код MetaPost и рисунок-результат

Если вы зададите вектор (u, v) и хотите найти точку на кривой с касательной, параллельной этому вектору, то нужен оператор `directiontime` — обратный к `direction of`. Оператор

```
directiontime (u,v) of p
```

возвращает числовое значение, что дает первое время t, когда путь имеет заданное направление. (Если такого времени нет, то результат будет -1.) Например, если `a` — это путь, нарисованный тонкой кривой на рис. 26.1, то `directiontime (1,1) of a` возвращает 0.2084.

Есть еще предопределенный макрос

```
directionpoint (u,v) of p,
```

который находит первую точку на пути, где есть данное направление. Макрос `directionpoint` производит сообщение об ошибке, если искомое направление отсутствует на пути.

Операторы `arclength` и `arctime of` относятся к времени на пути более знакомым образом, используя понятие длины дуги. Выражение

```
arclength  p
```

выдает длину дуги пути. Если `p` — путь и `a` — число между 0 и `arclength p`, то

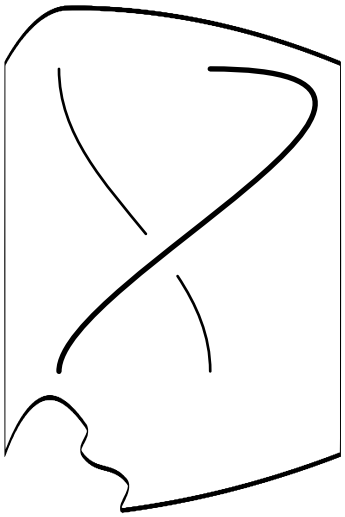
```
arctime a of p
```

дает время `t`, такое что `arclength subpath (0,t) of p = a`.

2. Примеры

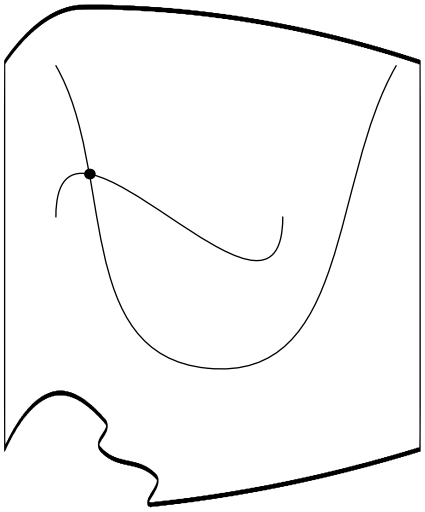
Пример 1

```
\mfpic[2]{0}{20}{0}{20}\mfsrc{u:=2cm;path p,q;
p:=(0,0){up}..(u,2u){dir 180};q:=(u,0){up}..(0,2u){up};
draw p withpen pencircle scaled 2bp;
draw subpath(0,.3) of q withpen pencircle scaled 1bp;
draw subpath(.45,1)of q withpen pencircle scaled 1bp;}
\endmfpic
```



Пример 2

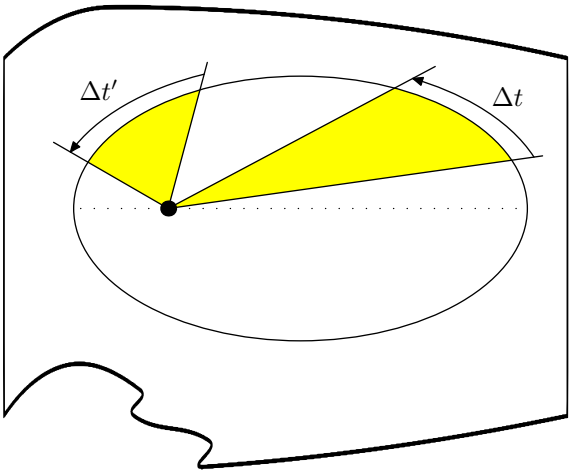
```
\mfpic[1]{0}{20}{0}{20}\mfsrc{path p, q;
p=(0,0){up} .. (3cm,0){up};
q=(0,2cm){dir -60}..(2cm,-2cm)..{dir 60}(4.5cm,2cm);
draw p;draw q;draw p intersectionpoint q withpen
pencircle scaled 4bp;   }
\endmfpic
```



Пример 3

```
\mfpic[1]{0}{100}{0}{100}%
\mfsrc{ u := 1cm; a := 6u; b := 3.5u;
 pair sun; sun := (-1.75u,0); path E[],
p[], area[];
 E1 = fullcircle xscaled a yscaled b;
  E2 = E1 scaled 1.1;
  p1 = sun -- (5u*dir(8) shifted sun);
   p2 = sun -- (4u*dir(28) shifted sun);
    p3 = sun -- (2u*dir(75) shifted sun);
     p4 = sun --(1.75u*dir(150) shifted sun);
      area1 = buildcycle(p1,E1,p2);
       area2 =buildcycle(p3,E1,p4);
 fill area1 withcolor red+green;
 fill area2 withcolor red+green;
  draw p1; draw p2; draw p3; draw p4; draw E1;
draw (-a/2,0)--(a/2,0) dashed withdots;
 draw sun withpen pencircle scaled 6bp;
 label.bot(btex Sun etex, sun);
numeric t[];
 t1 = ypart (p1 intersectiontimes E2);
 t2 = ypart (p2 intersectiontimes E2);
```

```
t3 = ypart (p3 intersectiontimes E2);
t4 = ypart (p4 intersectiontimes E2);
drawarrow subpath (t1,t2) of E2;
drawarrow subpath (t3,t4) of E2;
label.urt(btex $\Delta t$ etex,
point((t1+t2)/2) of E2);
label.ulft(btex $\Delta t'$ etex,
point((t3+t4)/2) of E2);} \endmfpic
```



Пример 4

```
\mfpic[.5]{0}{100}{0}{100}%
\mfsrc{path c;c=(0,0){dir(160)}..{dir(-100)}(100,50);
pair p[];p0 = point(0) of c; p1 = point(1) of c;
p2 = point(0.5) of c;        p3 = direction(0) of c;
p4 = direction(0.5) of c;    p5 = direction(1) of c;
dotlabel.lrt(btex $p0$ etex, p0);
dotlabel.lrt(btex $p1$ etex, p1);
dotlabel.lrt(btex $p2$ etex,p2);             draw c;
draw (-p3 -- p3) scaled (60/xpart(p3)) shifted p0;
draw (-p4 -- p4) scaled (60/xpart(p3)) shifted p2;
draw (-p5 -- p5) scaled (60/xpart(p3)) shifted p1;  }
 \endmfpic
```

Лекция 27

Стили рисования и модификация рисунков

Аффинные преобразования, заполнение рисунка, вырезание клиров.

1. Пунктирные линии

Язык MetaPost предоставляет много способов изменять линии помимо простого изменения их толщины. Один такой способ в использовании пунктирных линий. Синтаксис для этого —

```
draw p dashed образец пунктира,
```

где *образец пунктира* — это в действительности специальный тип `выражения-картинки`. Есть предопределенный *образец пунктира*, называемый `evenly`, делающий тире длиной 3 PostScript пункта, разделенные промежутками такого же размера. Другой предопределенный образец, `withdots`, производит точечную линию с точкой на каждые 5 PostScript пунктов. Для более разряженных точек и разряженных и удлиненных тире `образец пунктира` может быть масштабирован как показано на рис. 27.1.

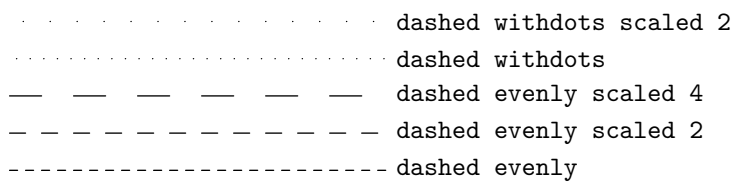


Рис. 27.1.

Пунктирные линии, помеченные `образцом пунктира`, используемым для их создания

Другой способ изменить образец линии в изменении его фазы горизонтальным сдвигом. Сдвиг вправо двигает образец вперед вдоль пути, а сдвиг вправо — назад. Рис. 27.2 иллюстрирует этот эффект.

Образец пунктира может быть указан более прямо через функцию `dashpattern`:

```
\texttt{dashpattern(on 6bp off 12bp on 6bp)}
```

Пример означает «нарисуй первые 6bp линии, затем пропусти следующие 12bp, затем рисуй 6bp и повторяй». Предопределенные образцы определены так:

```
evenly = dashpattern(on 3 off 3);
withdots = dashpattern(off 2.5 on 0 off 2.5);
```

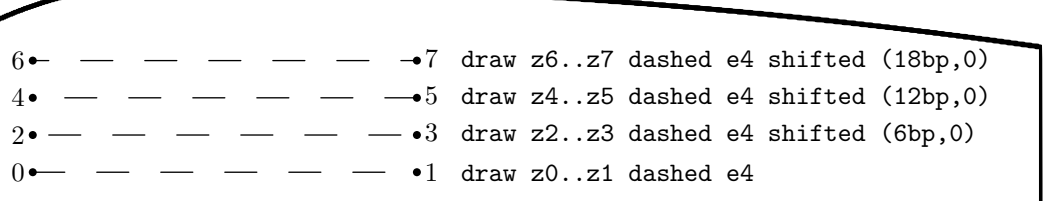


Рис. 27.2.

Пунктирные линии и команды MetaPost для их рисования, где `e4` ссылается на образец пунктира `evenly scaled 4`

2. Аффинные преобразования

Существует семь операторов трансформации с аргументом числом или парой :

$$
\begin{aligned}
(x,y)\ \texttt{shifted}\ (a,b) &= (x+a,\ y+b);\\
(x,y)\ \texttt{rotated}\ \theta &= (x\cos\theta - y\sin\theta,\ x\sin\theta + y\cos\theta);\\
(x,y)\ \texttt{slanted}\ a &= (x+ay,\ y);
\end{aligned}
$$

$$
\begin{aligned}
(x,y)\ \texttt{scaled}\ a &= (ax,\ ay);\\
(x,y)\ \texttt{xscaled}\ a &= (ax,\ y);\\
(x,y)\ \texttt{yscaled}\ a &= (x,\ ay);\\
(x,y)\ \texttt{zscaled}\ (a,b) &= (ax-by,\ bx+ay).
\end{aligned}
$$

Большинство этих операторов объясняют сами себя, за исключением `zscaled`, о котором можно думать как о произведении комплексных чисел. Эффект `zscaled` (a,b) — это поворот и масштабирование так, что $(1,0)$ переходит в (a,b). Эффект от `rotated` θ во вращении на θ градусов против часовой стрелки.

Любая комбинация сдвигов, вращений, наклонов и т. п. — это аффинная трансформация, совокупный эффект которой в трансформации любой пары (x,y) в

$$(t_x + t_{xx}x + t_{xy}y,\ t_y + t_{yx}x + t_{yy}y),$$

для некоторой шестерки $(t_x, t_y, t_{xx}, t_{xy}, t_{yx}, t_{yy})$. Последняя информация может быть сохранена в переменной типа трансформация, так что `transformed T` может быть эквивалентно

```
xscaled -1 rotated 90 shifted (1,1);
```

если `T` — это переменная-трансформация. Трансформация `T` могла быть инициализирована выражением типа трансформация, например,

```
transform T; T = identity xscaled -1 rotated 90
shifted (1,1);
```

Этот пример показывает, что выражения-трансформации могут строиться применением трансформационных операторов к другим трансформациям. Стандартная трансформация `identity` — это полезная отправная точка для этого процесса. Все можно проиллюстрировать переводом уравнения выше на естественный язык: “`T` следует быть трансформацией, такой же как `identity`, затем масштабированной по координатам x в -1 раз, вращенной на 90° и сдвинутой на $(1,1)$.” Это работает, потому что

`identity` — тождественная трансформация, которая ничего не делает, т. е. `transformed identity` — это пустой оператор.

Имеются две дополнительные возможности для **трансформации**:

```
reflectedabout (p, q)
```

отражает относительно прямой, определяемой точками p и q и

```
rotatedaround(p,\theta)
```

вращает на θ градусов против часовой стрелки вокруг точки p. Например, уравнением для инициализации трансформации `T` может быть

```
T = identity reflectedabout((2,0), (0,2))
```

Есть еще унарный оператор с одним аргументом `inverse`, находящий обратную трансформацию, отменяющую эффект трансформации-аргумента. Таким образом, если

$$p = q \texttt{ transformed } T, \qquad \text{то} \qquad q = p \texttt{ transformed inverse } T.$$

Нельзя брать `inverse` для неизвестной трансформации, но мы уже видели, что можно

```
T =  выражение-трансформация
```

когда `T` еще не имеет значения. Также возможно применять неизвестную трансформацию к известной паре или трансформации и использовать результат в линейном уравнении. Три таких уравнения достаточны для определения трансформации. Таким образом, уравнения

```
(0,1) transformed T' = (3,4); (1,1) transformed
T' = (7,1); (1,0) transformed T' = (4,-3);
```

позволяют MetaPost определить, что трансформация `T'` — это комбинация вращения и масштабирования с

$$\begin{aligned} t_{xx} &= 4, & t_{yx} &= -3, \\ t_{yx} &= 3, & t_{yy} &= 4, \\ t_x &= 0, & t_y &= 0. \end{aligned}$$

```
path p[]; p1 = fullcircle scaled .6in;
z1=(.75in,0)=-z3; z2=directionpoint left of p1=-z4;
p2 = z1..z2..{curl1}z3..z4..{curl 1}cycle;
fill p2 withcolor .4[white,black]; unfill p1; draw p1;
transform T; z1 transformed T = z2;
z3 transformed T = z4;xxpart T=yypart T;
yxpart T=-xypart T; picture pic;
pic = currentpicture; for i=1 upto 2:
pic:=pic transformed T; draw pic;
endfor dotlabels.top(1,2,3); dotlabels.bot(4);
```

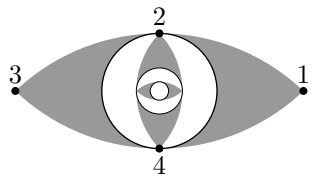


Рис. 27.3.
Код MetaPost и фрактальная картинка-результат

Уравнения с неизвестными трансформациями рассматриваются как линейные уравнения с шестью параметрами, определяющими трансформацию. Эти шесть параметров могут также именоваться напрямую как

`xpart T`, `ypart T`, `xxpart T`, `xypart T`, `yxpart T`, `yypart T`,

где `T` — это трансформация. Например, рис. 27.3 использует уравнения

```
xxpart T=yypart T; yxpart T=-xypart T
```

для указания, что `T` сохраняет форму, т. е. является комбинацией вращения, сдвига и сохраняющего форму масштабирования.

3. Стрелки

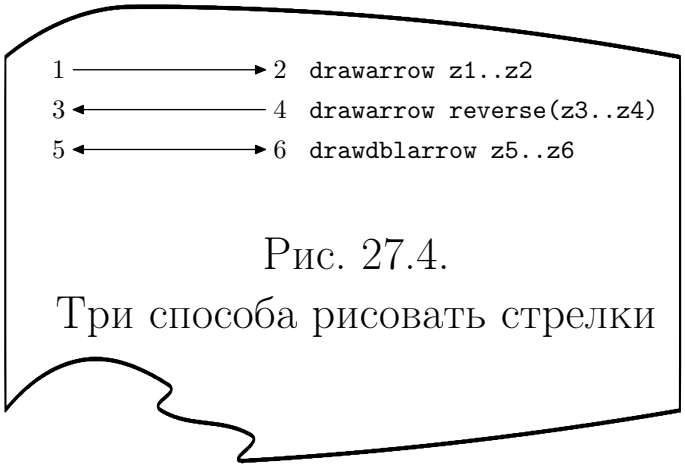


Рис. 27.4.
Три способа рисовать стрелки

Нарисованные стрелки, подобные тем, что на рис. 27.4, — это просто результат, когда вместо `draw` *выражение-путь* пишут

`drawarrow` *выражение-путь*.

Последнее рисует путь с наконечником стрелы в последней точке пути. Если вы хотите наконечник стрелы в начале пути, то просто используйте унарный оператор `reverse`, который из пути-аргумента делает обратный путь относительно времени, т. е. для пути `p` с `length p` $= n$,

`point` t `of reverse p` и `point` $n - t$ `of p`

— синонимы.

Как показано на рис. 27.4, команда, начинающаяся с

`drawdblarrow` *выражение-путь*,

рисует наконечники стрелок в обоих концах пути. Размер наконечника гарантируется большим, чем толщина линии, но возможно понадобится дополнительная настройка для очень толстых линий. Это делается присвоением нового значения внутренней переменной `ahlength`, которая определяет длину наконечника, как показано на рис. 27.5. Увеличение `ahlength` более стандартного значения 4 PostScript-пункта до 1.5 сантиметров производит большой наконечник на рис. 27.5. Есть еще параметр `ahangle`, контролирующий угол на верхушке наконечника стрелки. Типовое значение этого угла — 45 градусов, что показано на рисунке.

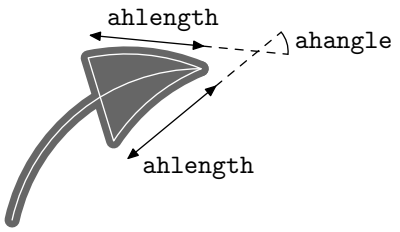


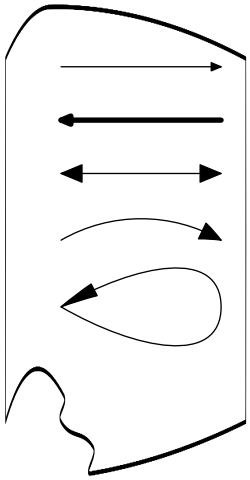
Рис. 27.5.
Большой наконечник стрелки с отмеченными ключевыми параметрами и с выделенными белыми линиями путями для его рисования

Приведем пример изменения параметра стрелки:

```
drawarrow (0,0)--(60,0);
drawarrow reverse((0,-20)--(60,-20))
withpen pencircle scaled 2;

ahlength:=0.3cm;
drawdblarrow (0,-40)--(60,-40);
drawarrow (0,-65){dir(30)}..
{dir(-30)}(60,-65);

ahlength:=0.5cm;
ahangle:=20;
drawarrow (0,-90){dir(-30)}..{up}(60,-90)..
{dir(-150)}cycle;
```

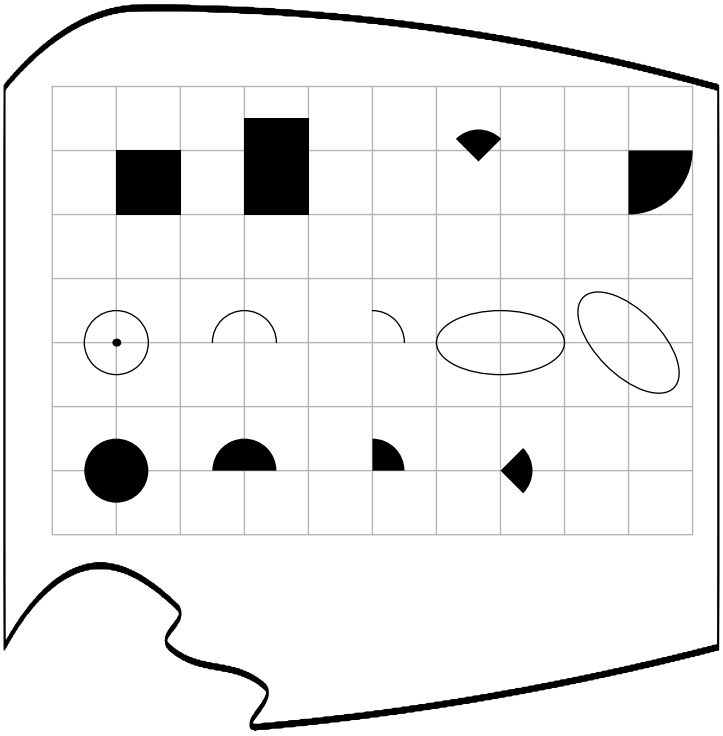


4. Стандартные кривые

Имеется некоторый стандартный набор кривых, которые можно использовать для рисования, применяя набор преобразований:

```
u := 24;
for i=-1 upto 9:
draw (i*u,4u)--(i*u,-3u) withcolor 0.7white;
endfor;
for i=-3 upto 4:
draw (-u,i*u)--(9u,i*u) withcolor 0.7white;
endfor;
dotlabel(,origin);
draw fullcircle scaled u;
draw halfcircle scaled u shifted (2u,0);
draw quartercircle scaled u shifted (4u,0);
draw fullcircle xscaled 2u yscaled u shifted (6u,0);
draw fullcircle xscaled 2u yscaled u
     rotated -45 shifted (8u,0);
fill fullcircle scaled u shifted (0,-2u);
fill halfcircle--cycle scaled u shifted (2u,-2u);
path quarterdisk; quarterdisk
     := quartercircle--origin--cycle;
fill quarterdisk scaled u shifted (4u,-2u);
```

```
fill quarterdisk scaled u rotated -45 shifted (6u,-2u);
fill quarterdisk scaled u shifted (6u,-2u) rotated 45;
fill quarterdisk rotated -90 scaled 2u shifted (8u,3u);
fill unitsquare scaled u shifted (0,2u);
fill unitsquare xscaled u yscaled 3/2u shifted (2u,2u);
```

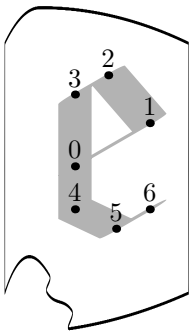


5. Перья

В уже рассмотренных примерах для рисования кривых командой `draw` применялись различные перья, которые задавались в виде

```
withpen   pencircle  scaled
```

и создавали линии указанной толщины. Отправной точкой для таких перьев служит круг диаметром один PostScript пункт. Толщина линии фиксируется только на момент рисования данной кривой. Фиксированное по размеру и форме перо можно при этом не только скалировать, но и подвергать линейным преобразованиям, что видно в многочисленных примерах. Аффинные трансформации производят круговую или эллиптическую форму пера. Толщина линии, рисуемой пером, зависит от того, насколько перпендикулярна линия большей оси эллипса. Это хорошо видно в следующем примере.



METAPOST имеет команду `pickup`, которая позволяет зафиксировать форму текущего пера для всего дальнейшего рисования. Приведем примеры.

```
pickup pencircle scaled 10;
pickup pensquare xscaled 20;
pickup pencircle xscaled 10 yscaled 35.
```

Исходное значение может быть также выбрано в любое время командой `pickup defaultpen`.

6. Построение циклов

Команда `buildcycle` конструирует пути для использования с макросами `fill` или `unfill`. Когда задаются два или более путей, таких как `aa` и `b`, макрос `buildcycle` пытается соединить их части вместе, формируя циклический путь. В рассмотренном ниже случае путь `aa` является полукругом, начинающимся справа от пересечения с путем `b`, затем проходящим через `b` и заканчивающимся снаружи круга слева, как показано на рис. 27.6a.

Рис. 27.6b показывает как `buildcycle` формирует замкнутый цикл из кусков путей `aa` и `b`. Макрос `buildcycle` находит два пересечения, помеченные 1 и 2 на рис. 27.6b. Затем он конструирует циклический путь, показанный выделенным на рисунке, двигаясь вдоль пути `aa` от пересечения 1 к пересечению 2 и затем против часовой стрелки по пути `b` обратно к пересечению 1. Кажется очевидным, что `buildcycle(a,b)` будет производить такой же результат, но основания для этого несколько путанные.

Проще всего использовать макрос `buildcycle` в ситуациях, подобных рис. 27.6, где есть более двух аргументов-путей и каждая пара последовательных путей имеет единственное пересечение.

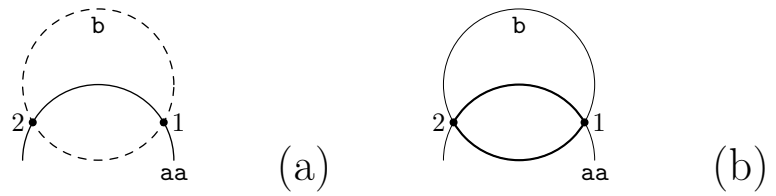


Рис. 27.6.
(a) Полукруговой путь `aa` с пунктирной линией, отмечающей путь `b`; (b) пути `aa` и `b` с частями, выделяемыми `buildcycle` и показанными жирными линиями

Например, прямая `q0.5` и кривая `p2` пересекаются только в точке P; кривая `p2` и прямая `q1.5` — только в точке Q. Фактически каждая из точек P, Q, R, S является единственным пересечением и результат команды

```
buildcycle(q0.5, p2, q1.5, p4)
```

берет `q0.5` от S до P, затем `p2` от P до Q, затем `q1.5` от Q до R и, наконец, `p4` от R обратно до S. Исследование кода MetaPost для рис. **??** открывает, что вы должны идти назад вдоль `p2` на переходе от P до Q. Все работает вполне совершенно до тех пор, пока точки пересечения определяются уникально, но может обусловить неожиданные результаты, когда пары путей пересекаются более одного раза.

Общее правило для макроса `buildcycle`:

$$\texttt{buildcycle(}p_1\texttt{,}\ p_2\texttt{,}\ p_3\texttt{,}\ \ldots\texttt{,}p_k\texttt{)}$$

выбирает пересечение между каждым p_i и p_{i+1} так, чтобы это было как можно дальше на p_i и как можно ближе на p_{i+1}. Первым находится пересечение между p_k и p_1, затем p_1 и p_2, $\ldots$ (прим. перев.).

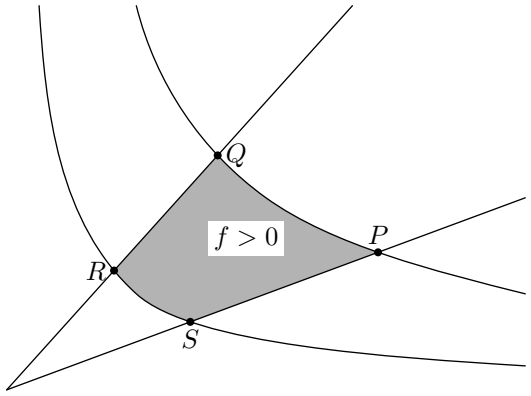


Рис. 27.7.
MetaPost-код и соответствующий вывод

```
beginfig(24);h=2in; w=2.7in;path p[], q[], pp;
for i=2 upto 4: ii:=i**2;
p[i]=(w/ii,h){1,-ii}...(w/i,h/i)...(w,h/ii){ii,-1};
endfor
q0.5=(0,0)--(w,0.5h);q1.5=(0,0)--(w/1.5,h);
pp = buildcycle(q0.5, p2, q1.5, p4);
fill pp withcolor .7white;z0=center pp;
picture lab; lab=thelabel(btex $f>0$ etex, z0);
unfill bbox lab; draw lab;
draw q0.5; draw p2; draw q1.5; draw p4;
dotlabel.top(btex $P$ etex,p2 intersectionpoint q0.5);
dotlabel.rt(btex $Q$ etex,p2 intersectionpoint q1.5);
dotlabel.lft(btex $R$ etex,p4 intersectionpoint q1.5);
dotlabel.bot(btex $S$ etex,p4 intersectionpoint q0.5);
endfig;
```

Нет простого правила для разрешения конфликтов между этими двумя целями, так что вам следует избегать случаев, когда одна точка пересечения случается дальше на p_i и другая точка пересечения случается ближе на p_{i+1}.

Установка на самые дальние пересечения для p_i и самые ближние для p_{i+1} ведет к устранения неясности предпочтением идущих впереди подпутей. Для циклических путей, как путь `b` на рис. 27.6, «близко» и «далеко» относительны по отношению к начальной/конечной точке, которая расположена там, куда вы попадаете обратно, сказав «`..cycle`». Для пути `b` эта точка устанавливается на самую правую точку на круге.

Лекция 28 ⏮

Программирование. Циклы и условные операторы. Макросы

Циклы и условные операторы. Макросы. Определение новых команд.

1. Макросы

MetaPost имеет множество автоматически включаемых макросов, называемое макропакет Plain, и некоторые команды определены Цель этого раздела объяснить, как писать такие макросы.

Макросы без аргументов очень просты. Определение макроса

```
def символический знак   = текст замены enddef
```

делает **символический знак** сокращением для **текста замены**, где **текст замены** может быть, в сущности, любой последовательностью знаков. Например:

```
def --- = ..tension infinity.. enddef.
```

Макросы с аргументами похожи, за исключением того, что они имеют формальные параметры, которые говорят, как использовать аргументы в **тексте замены**. Например, макрос `rotatedaround` определяется примерно так:

```
def rotatedaround(expr z, d) =
  shifted -z rotated d shifted z enddef;
```

и выражение

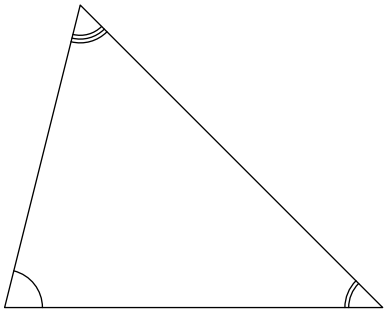
```
z1 rotatedaround (z2; 30)
```

просто раскрывается как

```
z1 shifted -z2 rotated 30 shifted z2.
```

Еще пример макроса с параметрами:

```
 def draw_angle(expr A, O, B, n) =
    draw_angle_(A,O,B,5mm);
    if n>1 : draw_angle_(A,O,B,4.5mm); fi;
    if n>2 : draw_angle_(A,O,B,4mm);   fi;
    if n>3 : draw_angle_(A,O,B,5.5mm); fi;
  enddef;
  def draw_angle_(expr A,O,B,d) =
    draw (O + d*unitvector(A-O))
         { d*unitvector(A-O) rotated 90 }
         ..
         { d*unitvector(B-O) rotated 90 }
         (O + d*unitvector(B-O));
  enddef;
  pair A, B, C;
A := (0,0);
B := (3cm,0);
C := (1cm,2cm);
draw A--B--C--cycle;  draw_angle(B,A,C,1);
draw_angle(C,B,A,2);  draw_angle(A,C,B,3);
```



Слово `expr` в определении значит, что формальные параметры `z` и `d` могут быть произвольными выражениями. Этим параметрам следует соответствовать выражениям-парам, но MetaPost не проверяет этого сразу.

Из-за того, что MetaPost — это интерпретируемый язык, макросы с аргументами очень похожи на подпрограммы. Макросы MetaPost часто

используются подобно подпрограммам, поэтому язык включает в себя программные концепции для такого использования. Эти концепции включают локальные переменные, циклы и условные команды.

1.1. Группировка

Группировка в MetaPost весьма важна для функций и локальных переменных. Базовая идея в том, что группа — это последовательность команд, возможно завершающаяся выражением, с обеспечением того, что некоторые символьные знаки могут восстановить свои старые значения в конце группы. Если группа заканчивается выражением, то группа ведет себя подобно вызову функции, что возвращает это выражение. В противном случае, группа — это просто составная команда. Синтаксис для группы

```
begingroup {список команд}  endgroup
```

или

```
begingroup {список команд}{выражение}  endgroup
```

где `{список команд}` — это последовательность команд, за каждой из которых следует точка с запятой.

Имена переменных, начинающиеся с `x` или `y` являются локальными в том смысле, что они считаются неизвестными в начале каждой картинки и их значения забываются в конце каждой картинки. Следующий пример иллюстрирует как работает локальность.

```
x23 = 3.1;
beginfig(17);
    ⋮
y3a=1; x23=2;
    ⋮
endfig;
show x23, y3a;
```

Результат команды `show`

```
>> 3.1
```

```
>> y3a
```

показывает, что `x23` возвращается к своему прежнему значению `3.1`, а `y3a` совершенно неизвестно, как оно и было в `beginfig(17)`.

2. Параметризованные макросы

Базовая идея макросов с параметрами в достижении большей гибкости передачей добавочной информации в макрос. Мы уже видели, что определения макросов могут иметь формальные параметры для выражений, задаваемых при вызове макроса. Например, определение

```
def rotatedaround(expr z, d)=текст замены  enddef
```

позволяет MetaPost-интерпретатору понимать вызовы макроса в форме

```
rotatedaround (выражение, выражение)
```

Ключевое слово `expr` в определении макроса значит, что параметры могут быть выражениями любого типа. Когда определение указывает `(expr z, d)`, то формальные параметры `z` и `d` ведут себя подобно переменным подходящего типа. Внутри *текста замены* они могут быть использованы в выражениях, как обычные переменные, но они не могут повторно декларироваться или получать новые значения. Нет ограничений на неизвестные или частично известные аргументы. Поэтому определение

```
def midpoint(expr a, b) = (.5[a,b]) enddef
```

прекрасно работает для неизвестных `a` и `b`. Уравнение, такое как

```
midpoint(z1,z2) = (1,1)
```

может быть использовано для определения `z1` и `z2`.

Заметьте, что определение выше для `midpoint` работает для чисел, пар или цветов — нужно только совпадение типов обоих параметров. Если зачем-то нужен макрос `middlepoint`, работающий только для пути или рисунка, то будет необходимо выполнять проверку `if` типа аргумента. Есть унарный оператор

`path` *первичность*,

возвращающий логический результат, показывающий является ли его аргумент путем. Из-за того, что проверка `if` имеет базовый синтаксис

`if` *логическое выражение*`:`
сбалансированные знаки `else:`
сбалансированные знаки `fi`,

где *сбалансированные знаки* могут быть чем-угодно, сбалансированным относительно `if` и `fi`, полный макрос `middlepoint` с проверкой типа может выглядеть так:

```
def middlepoint(expr a) = if path a:
                    (point .5*length a of a)
else: .5(llcorner a + urcorner a) fi enddef;
```

Позволяются многократные проверки `if`, например,

```
if e1: ... else: if e2: ... else: ... fi fi
```

которые можно сократить до

```
if e1: ... elseif e2: ... else: ... fi
```

где e_1 и e_2 представляют логические выражения.

Настоящее назначение макросов и проверок `if` в автоматизации решения повторяющихся задач, а также в возможности решать подзадачи по-отдельности.

3. Макросы vardef

Определение макроса может начинаться с `vardef` вместо `def`. Макрос, определенный таким образом, называется vardef-макросом. Они особенно хорошо подходят к приложениям, где макросы используются как функции или процедуры. Главная идея в том, что vardef-макрос подобен переменной типа “макрос”.

Вместо `def`*символический знак* vardef-макрос начинает

`vardef` *обобщенная переменная*,

где*обобщенная переменная* – это имя переменной с числовыми индексами, замененными на символ обобщенного индекса — `[]`. Другими словами, имя после `vardef` обязано иметь в точности такой же синтаксис как и имя в декларации переменной. Vardef-макрос ведет себя подобно процедуре или функции.

Суммируем особые свойства vardef-макросов: они допускают как широкий класс имен для макросов, так и имена макросов, за которыми следует специальный параметр-суффикс. Более того, `begingroup` и `endgroup` автоматически добавляются к *тексту замены* vardef-макроса.

Фактически, большинство определений макросов в примерах могут использовать `vardef` вместо `def`. Обычно не имеет большого значения, который из них использовать, но есть хорошее общее правило: используйте `vardef`, если вы собираетесь использовать макрос как функцию или процедуру. Следующее сравнение должно помочь при решении, когда использовать `vardef`.

- Vardef-макрос автоматически окружен `begingroup` и `endgroup`.
- Имя vardef-макроса может быть более одного знака длиной и оно может содержать индексы.
- Vardef-макрос может иметь доступ к суффиксу, следующему за именем макроса при его вызове.
- Когда символический знак используется в имени vardef-макроса, он может по-прежнему использоваться в именах других переменных.

4. Циклы

Многочисленные примеры в предыдущих разделах использовали простую форму цикла `for`,

`for` *символический знак* `=` *выражение* `upto` *выражение* : *тело цикла* `endfor`

Конструировать цикл с уменьшением также просто — нужно лишь заменить `upto` на `downto` и сделать второе *выражение* меньшим первого.

Этот раздел описывает более сложные ситуации: прогрессии; циклы, где параметр цикла ведет себя как суффикс; способы выхода из цикла.

Начнем с представления общего факта того, что `upto` — это предопределенный макрос, равный

`step 1 until`,

и `downto` — макрос, равный `step -1 until`. Цикл, начинающийся с

`for i=a step b until c`,

перебирает последовательность `i` из значений `a`, $a+b$, $a+2b$, ..., останавливаясь до того как `i` пройдет за `c`, т. е. цикл перебирает значения `i`, где $i \leqslant c$ при $b > 0$ и $i \geqslant c$ при $b < 0$. Для $b = 0$ цикл никогда не закончится даже при $a = c$.

Лучше всего использовать такой цикл только тогда, когда шаг цикла — это целое или число, представимое точно в виде дроби, кратной $\frac{1}{65536}$. В противном случае будет накапливаться ошибка и параметр цикла может не достичь ожидаемого конечного значения. Например,

```
for i=0 step .1 until 1: show i; endfor
```

покажет десять значений `i`, последнее из которых будет 0.90005.

Стандартный способ избежать проблемы, связанной с нецелым размером шага, в итерации по целым значениям и умножению их при использовании на масштабирующий множитель.

Есть альтернатива — можно указывать значения для перебора явно. Любая последовательность из нуля и более выражений, разделенных запятыми, может использоваться на месте `a step b upto c`. Причем, все выражения не обязаны иметь одинаковый тип или известные значения. Таким образом,

```
for t=3.14, 2.78, (a,2a), "hello": show t; endfor
```

покажет четыре значения из списка.

Заметьте, что тело цикла в примере выше — это команда, за которой следует точка с запятой. Для тела цикла естественно содержать одну

или более команду, хотя это и не обязательно. Цикл подобен определению макроса, за которым следует вызов этого макроса. Тело цикла может быть практически любой последовательностью знаков, имеющих смысл в совокупности. Поэтому (нелепая) команда

```
draw for p=(3,1),(6,2),(7,5),(4,6),(1,3):
p-- endfor cycle;
```

эквивалента

```
draw (3,1)--(6,2)--(7,5)--(4,6)--(1,3)--cycle;
```

Если цикл подобен определению макроса, то параметр цикла подобен `expr`-параметру. Он может представлять любое значение, но он не является переменной и его нельзя менять командой присваивания.

Когда нет значений для перебора, вы можете использовать цикл `forever`,

`forever:` *тело цикла* `endfor`

Для завершения такого цикла, когда логическое условие станет истинным, используйте пункт выхода:

`exitif` *логическое выражение*`;`

Когда интерпретатор MetaPost встречает пункт выхода, он вычисляет *логическое выражение* и выходит из текущего цикла, если оно истинно. Если более удобно выйти из цикла, когда выражение станет ложным, то используйте предопределенный макрос `exitunless`.

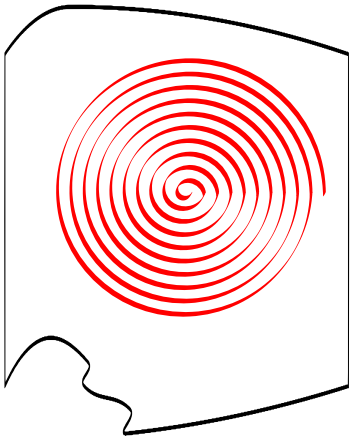
Поэтому вариант MetaPost для цикла **while** — это

`forever: exitunless` *логическое выражение*`;` *тело цикла* `endfor`

Пункт выхода может помещаться как непосредственно перед `endfor`, так и в любом другом месте *тела цикла*. Любой цикл может практически содержать любое количество пунктов выхода.

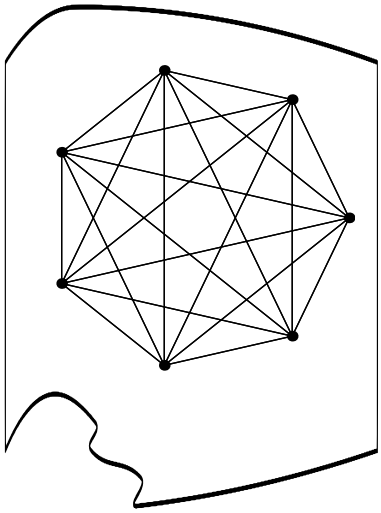
Пример 1

```
\mfpic[1]{0}{100}{0}{100}%
\mfsrc{pickup pencircle xscaled 2bp yscaled 0.25bp
rotated 60 withcolor red; for i=10 downto 1:
draw 5(i,0)..5(0,i)..5(-i,0)
..5(0,-i+1)..5(i-1,0) withcolor red;
      endfor;  }
\endmfpic
```



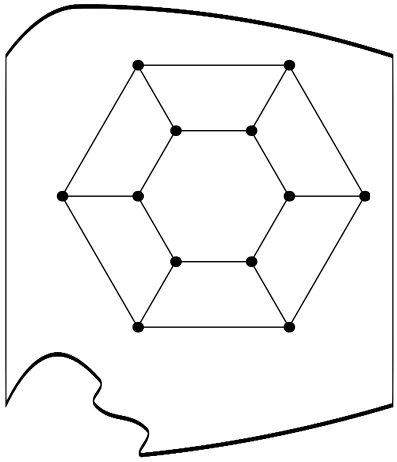
Пример 2

```
\mfpic[1]{0}{100}{0}{100}%
\mfsrc{u:=2cm;
pair A[]; numeric n; n:=7; A[0] = (u,0);
  for i=1 upto n-1:
     A[i] = A[0] rotated (360/n*i); endfor;
  for i=0 upto n-1:
    draw A[i] withpen pencircle scaled 4bp;
  for j=0 upto n-1:
if i<>j: draw A[i]--A[j] fi;
     endfor;
 endfor; }
 \endmfpic
```



Пример 3

```
\mfpic[1]{0}{100}{0}{100}\mfsrc{pair A[],B[];
numeric n; n:=6;  for i=0 upto n-1:
    A[i] = 1cm * right rotated (i*360/n);
    B[i] = 2cm * right rotated (i*360/n);
endfor; A[n]=A[0];B[n]=B[0];for i=0 upto n-1:
    draw A[i] -- A[i+1] -- B[i+1] -- B[i];
    draw A[i] withpen pencircle scaled 4bp;
    draw B[i] withpen pencircle scaled 4bp;
endfor;   }
\endmfpic
```



Лекция 29 ⏮

Взаимодействие METAPOST'а и LaTeX'а

О механизме взаимодействия METAPOST'а и LaTeX'а. Добавление текста к рисунку. Метки.

1. Интеграция текста и графики

MetaPost имеет несколько возможностей для включения меток и прочего текста в генерируемые им рисунки. Простейший способ сделать это в использовании команды `label`

```
label суффикс метки
      (выражение-строка или картинка выражение-пара);
```

Выражение-строка или картинка задает метку, а *выражение-пара* позицию для нее. *Суффикс метки* может быть **пустым**, что будет означать центрировать метку на заданных координатах. Если вы размечаете некоторые участки диаграммы, то вам вероятно понадобится слегка сместить метку, чтобы избежать накладки. Это иллюстрируется на рис. **??**, где метка `a` размещается над серединой указываемой линии, а метка `b` — слева от середины своей линии. Это достигается использованием `label.top` для метки `a` и `label.lft` для метки `b`, как показано на рисунке. **Суффикс метки** указывает позицию метки относительно заданных координат. Полное множество возможностей — это

$$\texttt{суффикс метки} \to \texttt{пусто} \mid \texttt{lft} \mid \texttt{rt} \mid \texttt{top} \mid \texttt{bot} \mid \texttt{ulft} \mid \texttt{urt} \mid \texttt{llft} \mid \texttt{lrt},$$

где `lft` и `rt` означают влево и вправо, а `llft`, `ulft` и т. п. значат вниз и влево, вверх и влево и т. п. Действительное расстояние, на которое будет смещена метка в заданном направлении, определяется внутренней переменной `labeloffset`.

Рис.29.1 также иллюстрирует команду `dotlabel`.

```
beginfig(17);
a=.7in; b=.5in;
z0=(0,0);
z1=-z3=(a,0);
z2=-z4=(0,b);
draw z1..z2..z3..z4..cycle;
draw z1--z0--z2;
label.top( a , .5[z0,z1]);
label.lft( b , .5[z0,z2]);
dotlabel.bot( (0,0) , z0);
endfig;
```

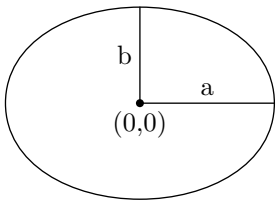


Рис. 29.1.
Код MetaPost и результат вывода

Она в точности такая же как команда `label`, за которой следует команда рисования точки в заданных координатах. Например,

```
dotlabel.bot( (0,0) , z0)
```

помещает точку в `z0` и затем размещает метку “(0,0)” точно под точкой.

Другой альтернативой является макрос `thelabel`. Он имеет такой же синтаксис, что и команды `label` и `dotlabel`, но он возвращает результат как **первичный рисунок** вместо его действительного изображения. Таким образом,

```
label.bot( (0,0) , z0)
```

эквивалентно

```
draw thelabel.bot( (0,0) , z0)
```

Для простых случаев размеченных рисунков, вам может обычно быть достаточно `label` и `dotlabel`. Дополнительно вы можете использовать короткую форму команды `dotlabel`, что сэкономит много времени, когда вы имеете много точек `z0`, `z1`, `z.a`, `z.b`, и т. п. и хотите использовать суффиксы `z` как метки. Команда

```
dotlabels.rt(0, 1, a);
```

эквивалентна

```
dotlabel.rt( 0 ,z0); dotlabel.rt( 1 ,z1);

dotlabel.rt( a ,z.a);
```

Таким образом, аргумент `dotlabels` — это список суффиксов для переменных `z`, а `суффикс метки`, задаваемый с `dotlabels`, используется для позиционирования всех меток.

Для команд разметки, таких как `label` и `dotlabel`, использующих строковые выражения для текста меток, строки печатаются в стандартном шрифте, определяемом строковой переменной `defaultfont`. Начальное значение `defaultfont` — это обычно `cmr10` , но оно может быть изменено на другое имя шрифта присваиванием, например,

```
defaultfont:= ptmr8r ,
```

`ptmr8r` — это типичный способ сослаться на шрифт Times-Roman в TEX.

Есть еще числовое количество, называемое `defaultscale`, определяющее размер шрифта. Пока `defaultscale` равно 1, вы получаете “нормальный размер”, который обычно равен 10 пунктам, но это можно изменить. Например,

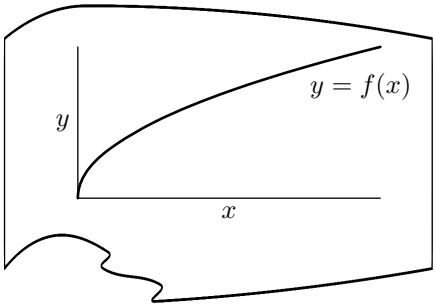
```
defaultscale := 1.2
```

делает метки на двадцать процентов больше. Если вам неизвестен нормальный размер и вы хотите быть уверенными в конкретном размере

шрифта, скажем 12 пунктов, вы можете использовать оператор `fontsize` для определения нормального размера, например,

```
defaultscale := 12pt/fontsize defaultfont;
```

Еще пример размещения меток:




```
numeric u;
u = 1cm;
draw (0,2u)--(0,0)--(4u,0);
pickup pencircle scaled 1pt;
draw (0,0){up}
for i=1 upto 8: ..(i/2,sqrt(i/2))*u endfor;
label.lrt(btex $y=f(x)$ etex, (3,sqrt 3)*u);
label.bot(btex $x$ etex, (2u,0));
label.lft(btex $y$ etex, (0,u));
```

Другие примеры меток можно посмотреть на приведенных в лекциях примерах.

2. Взаимодействие METAPOST'а и LaTeX'а

Возможны два способа использования графики в LaTeX'е. Первый — это использовать готовый файл с результатом обработки некоторого METAPOST-файла, который изготовлен вне системы LaTeX самостоятельно, либо взят из многочисленных источников разбросанных по различным сайтам в интернете. Тогда простейший файл для вставки графики, подготовленной METAPOST'ом и содержащейся в файле `foo.1`, в документ LaTeX'а может иметь вид:

```
\documentclass{article}
\usepackage{graphics}
\begin{document}
\includegraphics{foo.1}
\ end{document}
```

С использованием PDF с TEX и LATEX ситуация аналогичная, но надо добавить строку

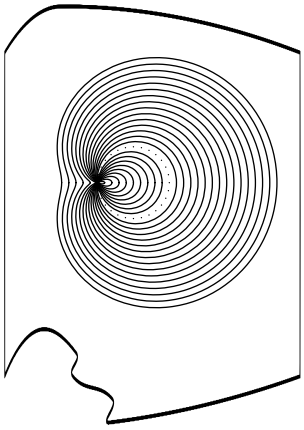
```
\DeclareGraphicsRule{*}{mps}{*}{}
```

к преамбуле документа после загрузки пакета graphicx. Эта декларация скажет pdfLATEX загружать все файлы с неизвестным расширением как mps-файлы — типовое расширение pdfLATEX для очищенных EPS-файлов

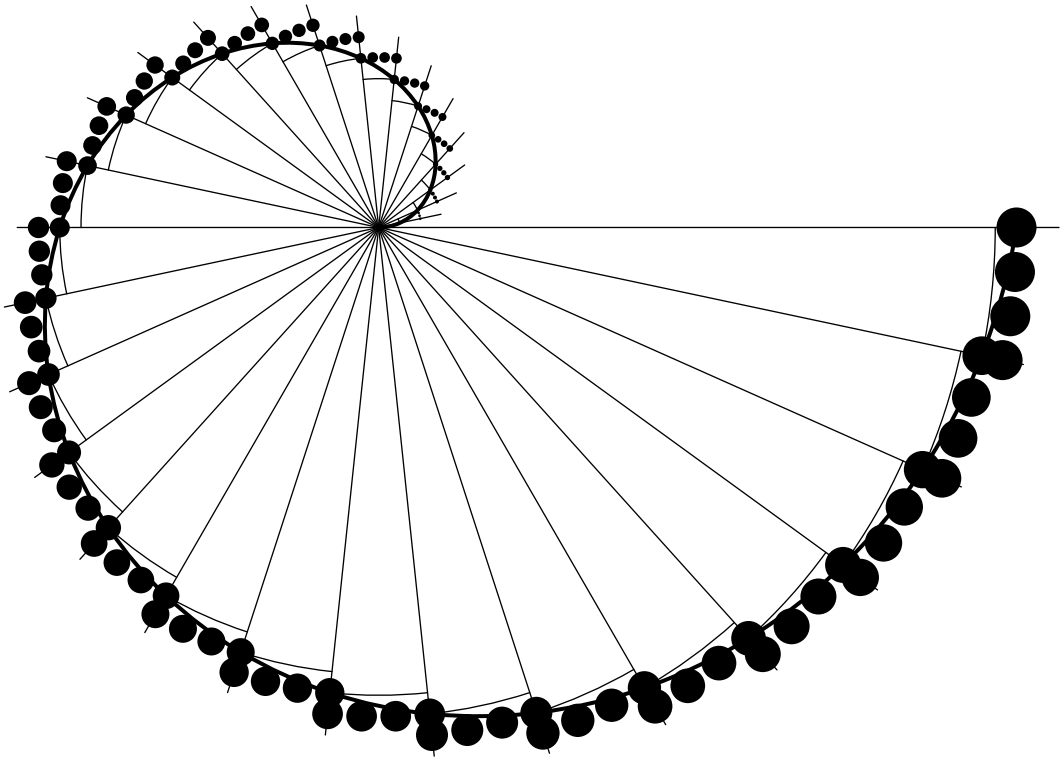
Либо дать универсальную декларацию, которая должна выглядеть подобно этой:

```
\usepackage{graphicx}
\usepackage{ifpdf}
\ifpdf
\DeclareGraphicsRule{*}{mps}{*}{}
\fi
```

При подготовке графики в недрах самого LATEX-файла с использованием METAPOST'а удобнее всего использовать пакет `mfpic` и команду `\mfsrc` для передачи данных в METAPOST. Наиболее простой способ — все графические построения проводить средствами пакета `mfpic`, а вычисления проводить METAPOST'ом передавая их через команду `\mfsrc`. Этим методом можно обойти сложность с выводом средствами METAPOST'а русскоязычного текста и сложных LATEX-выражений (например, математических конструкций) через метки. Посмотрите на двух приведенных ниже примерах этот прием.



```
\begin{mfpic}[50]{-3}{3}{-3}{4}
\mfsrc{for n=1 upto 15:}
 \plrfcn{-180,180,10}{cosd(t)+n/10}
\mfsrc{endfor;} \pen{1pt}
 \dotted\plrfcn{-90,90,10}{cosd(t)}
\end{mfpic}
```

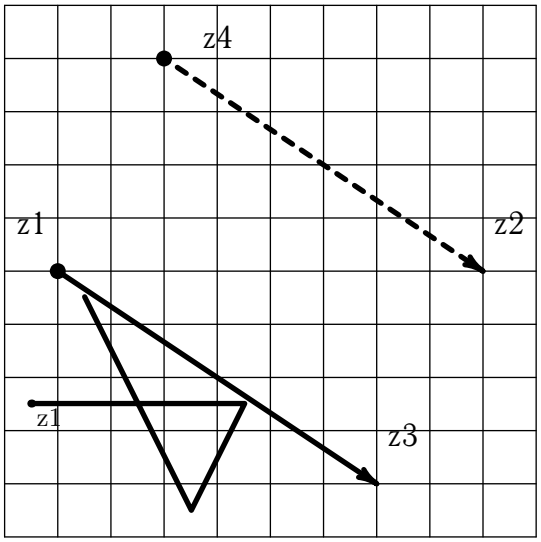


```
\begin{mfpic}[18]{-3.3}{6.5}{-5}{2.3}
\lines{(0,0),(cosd(12*1)*(0.2*1+0.4),
sind(12*1)*(0.2*1+0.4))}
\mfsrc{for n=2 upto 30:}\lines{(0,0),
(cosd(12*n)*(0.2*n+0.4),sind(12*n)*(0.2*n+0.4))}
\arc[p]{(0,0),12*(n-1),12*n,0.2*n-0.2}
\dotted[0.5*n,0.5*n]\arc[p]{(0,0),12*(n-1),12*n,0.2*n}
\mfsrc{endfor;}\pen{1.5pt}\plrfcn[s]{0,360,12}{t/60}
\ end{mfpic}
```

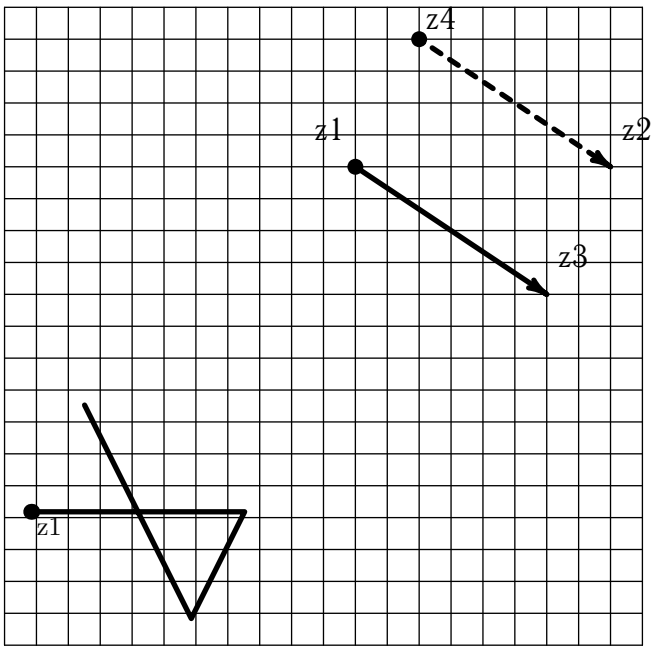
При данном подходе надо внимательно следить за единицами измерения, которые используются пакетом `mfpic` и METAPOST'ом. По умолчанию они задаются в пунктах и расхождения появятся, если мы, например, изменим только `\mfpicunit`.

Аналогичные предупреждения необходимо сделать о системах координат, которые используются пакетом `mfpic` и METAPOST'ом. Когда мы меняем границы рисунка в пакете `mfpic` он переходит от абсолютной системы координат к относительной и рисование командами `mfpic`'а и METAPOST'а приводит к разным результатам. Это видно на приведенных ниже примерах, где вектора и линия проведены в разных средах.

```
\begin{mfpic}[1]{0}{100}{0}{100}
\gridlines{10,10} \pointsize=6pt \pen{2pt}
\mfsrc{z1=(10,50);z2=(90,50);z3=(70,10);z3-z1=z2-z4;
dotlabel.lrt( z1 , z1);draw z1--z2--z3--z4;}
\arrow\lines{z1,z3}\arrow\dashed\lines{z4,z2}
\point{z1} \fillcolor{Red}\point{z4}
\tlabels{[br] (10,55)
{{\small z1}} [bl] (90,55) {\small z2} [bl]
(70,15) {\small z3} [br] (45,90) {{\small z4}}}
\end{mfpic}
```




```
\begin{mfpic}[1]{-100}{100}{-100}{100}
\gridlines{10,10}\pointsize=6pt \pen{2pt}
\mfsrc{z1=(10,50);z2=(90,50);z3=(70,10);z3-z1=z2-z4;
dotlabel.lrt( z1 , z1)withpen pencircle scaled 6pt;
draw z1--z2--z3--z4;}%
\arrow\lines{z1,z3}\arrow\dashed\lines{z4,z2}%
\point{z1} \fillcolor{Red}\point{z4}%
\tlabels{[br](10,55){{\small z1}}[bl](90,55){\small
z2}[bl](70,15) {\small z3} [br] (45,90) {{\small z4}}}
\end{mfpic}
```



Лекция 30

3D графика и мультипликация

Расширение возможностей METAPOSTза счет использования интерпретатора PERL в пакете трехмерной графики mfpic3d. Пакет movie15: мультипликация и PdfLaTeX

1. Поддержка трехмерной графики в LaTeX

В данной лекции, прежде всего, рассмотрим некоторые возможности пакета трехмерной иллюстративной графики `mfpic3d`, разработанного два года назад С.В.Знаменским на базе `mfpic` и `perl`, и уже эффективно применявшегося для подготовки к изданию книг по аналитической и дифференциальной геометрии. В настоящее время пакет активно совершенствуется. Эти картинки изготовлены с помощью `mfpic3d`:

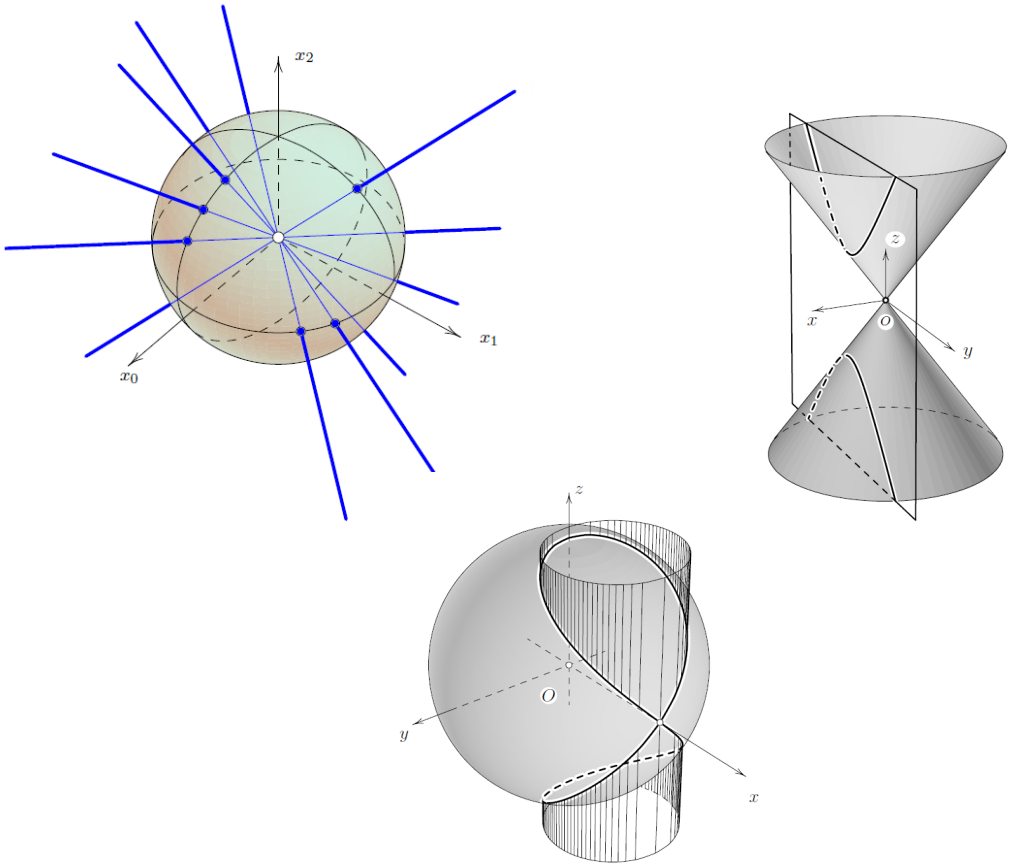


Рис 30.1. Примеры изображений в `mfpic3d`

Пакет `mfpic3d` использует интерпретатор `PERL` для расширения возможностей METAPOST, превращая его в уникальный инструмент для ри-

сования трехмерных геометрических объектов — кривых, поверхностей и их фрагментов с высоким качеством отработки пересечений. Являясь расширением пакета METAPOST, пакет `mfpic3d` предназначен для создания высококачественной трехмерной графики, импортируемой в TEX и LATEX документы.

1.1. Установка mfpic3d

Комплектация

Для того, чтобы воспользоваться возможностями пакета, на компьютере должен быть установлен MiKTEX версии не ниже шестой. В редактируемом файле `имяфайла.tex` с расширением `.tex` должен быть подключен пакет

```
\usepackage[metapost,truebbox,mplabels,dvips]{mfpic}
```

В преамбуле `имяфайла.tex` необходимо подключить стилевой файл `mfpic3d.sty`. Поскольку `mfpic3d` использует интерпретатор `PERL`, на компьютере должен быть установлен `PERL`, а файл `mfpic3d.pl` необходимо поместить в директорию, в которой TEX сможет его найти (например, в текущую рабочую директорию).

Для компиляции Файла необходим `FAR` или любой другой редактор, позволяющий работать с командной строкой.

Последовательность обработки

Обрабатывается, как обычно, основной файл `имяфайла.tex`, для этого в командной строке надо дать команду `latex имяфайла.tex`. После этого в рабочей директории появятся файлы: `mp1.mp , mp2.mp , ...` Их столько, сколько картинок содержится в файле `имяфайла.tex`.

Далее необходимо обработать эти файлы так, чтобы они стали «читабельны» для команды `mp.exe`. Для этого

- в командной строке дается команда: путь файла `perl.exe mfpic3d.pl mp1.mp` (стандартное расположение: `C:\PERL\bin\perl.exe mfpic3d.pl mp1.mp`), после чего в директории появится новый файл `mp1.mp.spl`;

- в командной строке дается команда: `makempx.exe mp1.mp mp1.mpx` (это стандартная процедура для обработки надписей `mplabels`), после чего в директории появится новый файл `mp1.mpx`.

Описанную процедуру необходимо проделать для всех файлов с расширением `*.mp`.

После этого Файлы `*.mp` готовы к обработке при помощи `mp.exe`. Как обычно при обработке картинок, в командной строке дается команда: `mp mp1.mp` (процедуру необходимо проделать для всех файлов `*.mp`).

В результате применения последней команды в рабочей директории создадутся файлы `mp1.1 , mp2.1 , ....` Это и есть картинки.

Для того, чтобы вставить их в итоговый файл, он обрабатывается, как обычно, файл `имяфайла.tex` LaTeX'ом, после чего в директории появится готовый, с картинками, файл с расширением `.dvi`.

Команда `dvips имяфайла.dvi` конвертирует `dvi`-файл в `ps`-файл, а команда `dvipdfm имяфайла.dvi`, если необходимо, сделает `pdf`-файл.

Заметим, что каждый раз обрабатывать таким способом картинки не требуется, если, конечно не возникает необходимости их изменять. Если все картинки уже сделаны, чтобы изменять текст файла `имяфайла.tex`, достаточно просто обработать его LaTeX'ом.

1.2. Специальные команды в mfpic3d

Для изображения пространственной кривой пакет `mfpic3d` определяет команды, позволяющие пользователю задавать пространственные координаты трехмерного объекта: (x,y,z). Основной принцип работы макрокоманд заключается в том, что задаваемые пользователем трехмерные координаты точки (x,y,z) пересчитываются `PERL`-ом в обычные двумерные, которые затем обрабатываются METAPOST-ом.

Картинка в стиле `mfpic3d` создается в окружении

```
\begin{mfpic} ... \end{mfpic}.
```

Все команды METAPOSTработают и в `mfpic3d`. Остановимся подробнее на специфических командах `mfpic3d`.

После `\begin{mfpic}` в окружении должны размещаться команды, которые обращаются к языку программирования `PERL`:

```
\perlvar inDomain  и \inDomain{1}.
```

Чтобы нарисовать линию или поставить точку в пространстве на картинке используют те же команды, что и для двумерного изображения, только задаются три координаты точки и перед каждой скобкой ставится косая черта `\(x,y,z\)`.

Точка зрения

Далее необходимо задать точку зрения, т. е. угол под которым будет видна трехмерная картинка. Для этого служит команда

```
\ViewPoint \{a,b,c\}.
```

Например, если задать параметры a, b, c, равные, соответственно, 5, 0 и 0, то наблюдатель будет помещен на ось Ox в точку с координатами (5,0,0). Пример изменения точки зрения приведен на рис. 30.2.

Приведем некоторые картинки, изготовленные с помощью `mfpic3d`.

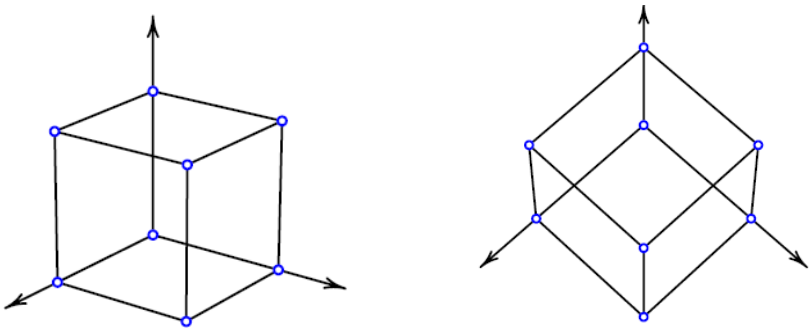


Рис 30.2 Куб: две точки зрения

Для первого куба точка зрения была задана командой `\ViewPoint {30,40,20}`, а для второго, соответственно, командой `\ViewPoint {20,20,50}`. Конечно, такой куб возможно нарисовать и средствами METAPOST, но пакет `mfpic3d` дает возможность изменять угол зрения на объект, меняя лишь аргументы команды `\ViewPoint`, тогда как в METAPOSTдля этого придется рисовать новый куб целиком. Остальные команды для рисования обоих кубов:

```
\mfsrc{z1=\(0,0,0\);z2=\(4,0,0\);z3=\(4,4,0\);
z4=\(0,4,0\);z5=\(0,0,4\);z6=\(4,0,4\);z7=\(0,4,4\);
z8=\(4,4,4\);z9=\(6,0,0\);z10=\(0,6,0\);z11=\(0,0,6\);}
\lines{z1,z2,z3,z4,z1,z5,z6,z8,z7,z5}\lines{z4,z7}
\lines{z8,z3}\lines{z6,z2}\arrow\lines{z1,z9}
\arrow\lines{z1,z10}\arrow\lines{z1,z11}
```

Изображение двумерной поверхности

Для того чтобы нарисовать изображение двумерной поверхности в пространстве, служит команда `\Definition{x(T), y(T), z(S)}`

```
\Definition{x(T), y(T), z(S)}
```

Ее аргументы `x(T), y(T), z(S)` — параметрические функции параметров `T` и `S`, задающие параметризацию поверхности (в цилиндрических координатах).

Например, команда `\Definition{x(T), y(T), z(S)}`, рисующая цилиндр, запишется в виде `\Definition{4*sin(T),4*cos(T),S}`.

Далее необходимо задать пределы, в которых изменяются `T` и `S`, для этого используют команды

```
\Tfrom{t1}\Tto{t2}\Sfrom{s1}\Sto{s2}
```

Команды `\Tsteps{n}` и `\Ssteps{k}` задают шаг, где значения `n` и `k` лучше брать равным степеням двоек, причем `n` намного больше `k`. Чем больше шаг, тем «глаже» изображение поверхности, но тем дольше `PERL` обрабатывает картинку. Ограничение сверху на `n` и `k` задается лишь возможностями PERLa и вашего компьютера.

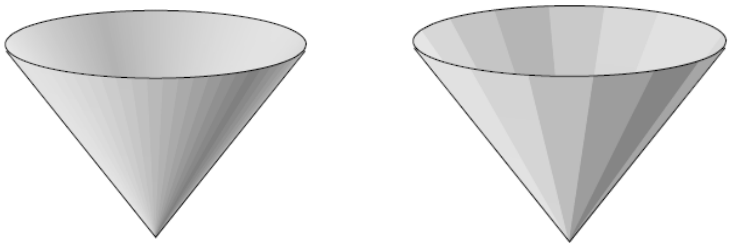


Рис 30.3. Сглаживание поверхности конуса

Команды `\Tpenwidth{Tstep}` и `\Spenwidth{Sstep}` задают жирность.

Команда `\Back` прорисовывает заднюю стенку (изнанку) поверхности. Поскольку на заднюю стенку поверхности накладывается изображение передней стенки поверхности, то нет надобности рисовать ее всю, достаточно задать отдельно пределы для параметров `T` и `S` и нарисовать только видимую часть. Например: `\Face\Tfrom{-(pi)}\Tto{pi}\Sfrom{-3}\Sto{3}`.

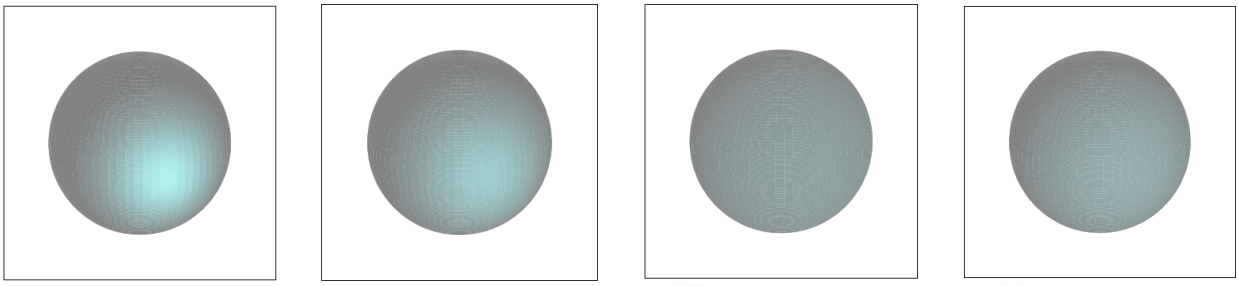
Команда `\Face` прорисовывает переднюю стенку поверхности в заданных пределах по `T` и `S` (см. например рис. 30.6).

Подсветка поверхности

Двумерная поверхность (сфера, цилиндр, конус) подсвечивается, создается светотень, для этого нужно использовать команду

```
\Lamps{[x, y, z], [d, e, f]}
```

Параметры `x, y, z` задают пространственные координаты точки, где помещена лампа, а `d,e` и `f` отвечают за яркость цвета: `d` — красного, `e` — зеленого, `f` — синего.



Lamps [15,-15,0],[.2,.5,.5] Lamps [20,-20,0],[.2,.5,.5] Lamps [200,-200,0],[.2,.5,.5] Lamps [40,-40,0],[.2,.5,.5]

Рис 30.4. Подсветка поверхности сферы

Цвет поверхности

Для того чтобы выбрать цвет поверхности, используют команду

```
\SurfaceColors {[a, b, c],[a, b, c]},
```

в которой первая тройка чисел задает раскраску передней, лицевой части поверхности (лица), а вторая — внутренней, изнаночной части поверхно-

сти (изнанки). Здесь параметры `a`, `b` и `c` определяют, соответственно, интенсивность зеленого, красного и желтого цветов.

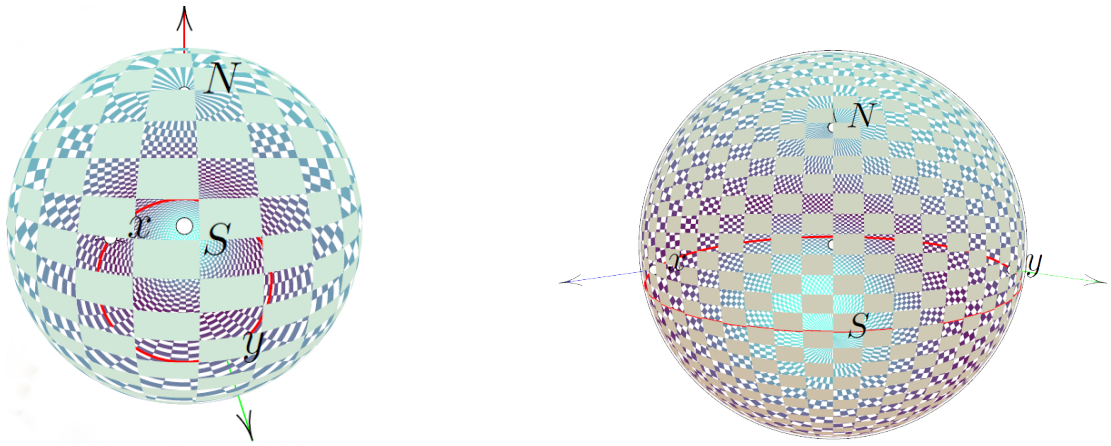


Рис 30.5. Подсветка и раскраска внутренней поверхности сферы

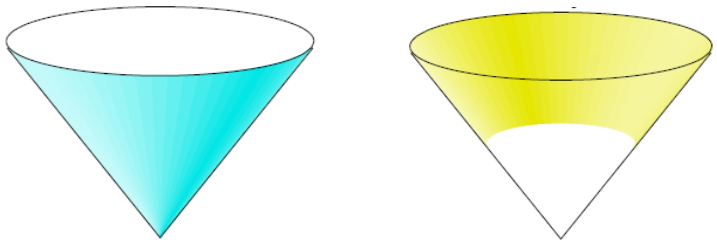


Рис 30.6. Подсветка поверхности конуса

Например, получить синюю раскраску лицевой поверхности конуса и желтую раскраску изнанки конуса можно следующим образом:

```
\SurfaceColors{[1, .1, .1], [.3, .3, .3]}
\Tfrom{.(pi)} \Tto{pi} \Sfrom{0} \Sto{6}
\tcaption{Конус: лицо}

\SurfaceColors{[.5, .5, .5], [.1, .1, 1]}
\Tfrom{.(pi)} \Tto{pi} \Sfrom{3} \Sto{6}
\tcaption{Конус: изнанка}
```

2. LaTeX и мультипликация

В заключение хочется представить, пожалуй, один из самых современных пакетов в LaTeX 2_ε и PdfLaTeX — пакет `movie15`, который получил некоторую известность только в 2007 г. Автор пакета Alexander Grahn.

Этот пакет предназначен создания анимированных pdf-файлов, в частности, для включения в pdf-файл средствами LaTeX'а видео, аудио и 3D объектов.

Для просмотра требуется Adobe Reader версии не ниже 6.

Пакет подключается, как обычно, в преамбуле документа при помощи команды

```
\usepackage[<package options>]{movie15}.
```

Опция «3D» дает доступ к современным возможностям включения 3D в PDF-1.6. При наличии необязательного аргумента «draft» медиафайлы не включаются в документ, вместо этого в прямоугольнике размеров <длина>x<высота>+<глубина> печатается имя включаемого файла. С другими опциями можно ознакомиться из подробного (англоязычного) описания пакета в файле movie15.pdf на http://www.ctan.org.

Медиафайл включается в PDF при помощи команды

```
\includemovie[<options>]{<width>}%
{<height>}{<media file>}
```

Здесь <width> (ширина) and <height> (длина) — необязательные аргументы, которые измеряются в единицах TeX'а. Вместо <media file> указывается имя медиафайла с расширением. Имя должно содержать только латинские буквы (и не содержать дополнительных символов, например "-").

Возможности пакета позволяют легко вставить в pdf-файл Flash-ролик в форматах .avi, .swf, а также видео в форматах, приведенных далее.

Рассмотрим пример включения файла в формате .avi. Команда \includemovie включения видео задается подобно команде \includegraphics с обязательными и необязательными аргументами:

```
{\includemovie[poster, text=(piramid.avi),%
mouse, repeat]%
{\linewidth}{.75\linewidth}{piramid.avi}}
```

Видеоклип запускается и останавливается по щелчку мыши:

(piramid.avi)

Пример включения `Flash`-ролика формате `.swf` (тест по математике):

(testmath.swf)

```
{\includemovie[poster,text=(testmath.swf),mouse,%
default]{\linewidth}{.98\linewidth}{testmath.swf}}
```

Аргумент `default` означает отсутствие внешнего управления роликом и позволяет выполнять тест по математике в соответствии с функциями самого `Flash`-ролика.

Возможности пакета позволяют также включать в `pdf`-файл MPEG видео, загружая его из Интернета, например с http://www.linux-video.net/Samples/. Естественно, в этом случае требуется рабочее подключение к Интернет.

(AlienSong.mpg)

Slow Normal Fast Play/Pause Stop

Заметим, что в отличие от предыдущего примера, в этом появляется возможность управлять видео при помощи кнопок управления, определяемых командой

```
\movieref[<options>]{<label spec>}{<text>}
```

Названия кнопкам <text>, в принципе, можно дать произвольные, но только латинскими буквами (пакет, к сожалению, пока не русифицирован). Аргумент <options> задает характер действия с файлом (останавливает, замедляет/ускоряет прокрутку и т. д.). Содержимое LaTeX'овского файла для нашего примера такое:

```
\includemovie[poster,label=alien,text=(AlienSong.mpg),
repeat,playerid=AAPL_QuickTime,url]{\linewidth}
{.7\linewidth}{http://www.linux-video.net/
Samples/Mpeg1/AlienSong.mpg}\\
```

Заметим, что вместо опции `url` на самом деле написана опция `inline=false` для того, чтобы блокировать обращение к интернет.

```
\ movieref[rate=0.5]{alien}{Slow}\hfill
\ movieref[default]{alien}{Normal}\hfill
\ movieref[rate=2]{alien}{Fast}\hfill
\ movieref[pause]{alien}{Play/Pause}\hfill
\ movieref[stop]{alien}{Stop}\vspace{19 ex}
```

И, наконец, последний пример — вложенный U3D файл. Он основывается на VRML модели, созданной Peter Whitehouse, http://www.wonko.info/vrml/index.htm. Конвертирование в U3D было выполнено при помощи DeepExploration®. Файл dice.vws позволяет управлять кубиком как при помощи мыши, так с использованием панели 3D инструментов. Для создания этого примера использовался язык Java.

(dice.u3d)

Click here to cycle through the list of predefined views. This link calculates 3D option settings for a 60° aperture angle of the virtual camera.

В заключение перечислим медиаформаты, которые могут быть включены в файл и воспроизводятся при помощи Adobe Reader:

Расширение	тип	Описание
aif, aifc, aiff	audio/aiff	Audio Interchange File Format
au, snd	audio/basic	NeXT/Sun Audio Format
avi	video/avi	Audio/Video Interleaved
gif	image/gif	(animated) GIF
mid, rmi	audio/midi	Musical Instrument Digital Interface
mov, qt	video/quicktime	Apple QuickTime
mp3, m3u	audio/x-mp3	MPEG Audio Layer-3
mp4	video/mp4	MPEG-4 Video
mpeg, mpg	video/mpeg	MPEG-1 & 2 Video
prc	model/prc	Adobe PRC Format (3D), requires Reader-8.1 or higher
ra	audio/vnd.rn-realaudio	Real Audio
rm, rv	application/vnd.rn-realmedia	Real Media (video & sound)
smil	application/smil	Synchronized Multimedia Integration Language
swf	application/x-shockwave-flash	Macromedia Flash
u3d	model/u3d	Universal 3D File Format
wav	audio/wav	MS Audio Format
wma	audio/x-ms-wma	Windows Media Audio
wmv	video/x-ms-wmv	Windows Media Video

Литература

[1] Грэтцер Г. *Первые шаги в LaTeX'e*/ Г.Грэтцер М.:Мир, 2000. - 172с.

[2] Гуссенс М. *Путеводитель по пакету LaTeXи его расширению LaTeX 2$_\varepsilon$*/ М.Гуссенс, Ф.Миттельбах, А.Самарин М.:Мир, 1999.

[3] Гуссенс М. *Путеводитель по пакету LaTeX и его Web-приложениям*/ М.Гуссенс, С.Ратц М.:Мир, 2001.- 604 с.

[4] Гуссенс М. *Путеводитель по пакету LaTeX и его графическим расширениям. Иллюстрирование документов при помощи TeX'a и PostScript'a*/ М.Гуссенс, С.Ратц, Ф.Миттельбах М.:Мир: Бином ЛЗ, 2002. - 621с.

[5] Кнут Д. Все про TeX/ Кнут Д. Протвино: АО RDTeX, 575 с., 1993. Тираж 10000 экз.

[6] Котельников И.А. ЛаТеХ по-русски/ Котельников И.А., Чеботаев П.З. Новосибирск: Сибирский хронограф, 2004. — 489 с.

[7] Львовский С. «Набор и верстка в пакете LaTeX»/ Львовский С. — М.: МЦНМО, 3-е издание. 2003.— 448 с.