LaTeX, GNU/Linux и русский стиль.

© Е.М. Балдин*

Эта статья была опубликована в январском номере русскоязычного журнала Linux Format (`http://www.linuxformat.ru`) за 2007 год. Статья размещена с разрешения редакции журнала на сайте `http://www.inp.nsk.su/~baldin/` и до июня месяца все вопросы с размещением статьи в других местах следует решать с редакцией Linux Format. Затем все права на текст возвращаются ко мне.

Текст, представленный здесь, не является точной копией статьи в журнале. Текущий текст в отличии от журнального варианта корректор не просматривал. Все вопросы по содержанию, а так же замечания и предложения следует задавать мне по электронной почте `mailto:E.M.Baldin@inp.nsk.su`.

Текст на текущий момент является просто *текстом*, а не книгой. Поэтому результирующая доводка в целях улучшения восприятия текста не проводилась.

*e-mail: E.M.Baldin@inp.nsk.su

Эмблемы TeX и METAFONT, созданные Дуайном Бибби, взяты со странички Д.Э. Кнута. Цветной пингвин взят из пакета `ps2pdf` от Ральфа Найпрашека (Rolf Niepraschk)

Оглавление

Глава 5

Документация и программный код

+++ Ошибка Деления На Огурец.
Переустановите Вселенную И Перезагрузитесь +++

Так зависает Гекс.
Источник: «Санта-Хрякус» от Терри Пратчетта

Программирование под Linux вполне естественное занятие. Написание документации неотъемлемая часть этого процесса. LaTeX достоин быть включённым в технологическую цепочку по выпуску программного продукта.

Если вспомнить историю, то Д.Э. Кнут создал TeX именно для целей представления кода и алгоритмов в своём глобальном пятитомнике «Искусство программирования».

5.1. Спецсредства

Чтобы украсить инструкцию надо добавлять в неё «пятна». Перегружать не стоит, но пару мыслей выделить вполне реально. Упомянутые ниже приёмы далеко не все имеющиеся в LaTeX — это просто демонстрация возможностей.

5.1.1. keystroke

Иногда в тексте необходимы фразы вида: «Для выхода из программы нужно нажать клавишу Esc.» Макрос \keystroke, определённый в одноимённом пакете **keystroke**, позволяет выделить название клавиши, примерно следующим образом:

```
Для продолжения нажмите
\keystroke{<<любую клавишу>>}.
```

Для продолжения нажмите «любую клавишу» .

В пакете определены так же многие клавиши, имеющиеся на стандартной клави-

атуре. Пакет очень прост и имеет зачатки интернационализации — его легко адаптировать для своих нужд.

\Spacebar		\Enter	Enter	\Return	↵
\Esc	Esc	\BSpace	⟵	\Tab	⟷
\Alt	Alt	\AltGr	AltGr	\Del	Del
\Shift	Shift ⇑	\PgUp	Page ↑	\PgDown	Page ↓
\End	End	\Ctrl	Ctrl	\Home	Home
\Ins	Ins	\LArrow	←	\RArrow	→
\UArrow	↑	\DArrow	↓	\PrtSc	PrtSc
\Scroll	Scroll	\Break	Break	\NumLock	Num
\keystroke{A}	A	\keystroke{Я}	Я	\keystroke{F1}	F1

Рис. 5.1. Клавиши, определённые в keystoke

5.1.2. LCD-дисплей

LCD-дисплеи сейчас встроены даже в кофемолки. Они легко узнаваемы, поэтому нет необходимости копировать их вид в документацию с помощью фотографий — достаточно нарисовать что-то похожее. Изобразить вид дисплея можно с помощью пакета LaTeX **lcd**.

```
\definecolor{darkgreen}{rgb}{0.22,0.26,0.19}
\definecolor{lightgreen}{rgb}{0.05,0.97,0.55}
\LCDcolors{darkgreen}{lightgreen}
\centering
\LARGE\textLCD{12}|Linux Format|\\[2mm]
\LCDcolors{lightgreen}{darkgreen}
\small\textLCD{12}|Linux Format|
```

Linux Format

Linux Format

Для определения цветов используется макрос \definecolor из пакета **color**. Команда \LCDcolors формирует цвет букв и фона, а макрос \textLCD выводит LCD-подобный текст на экран. \textLCD понимает стандартные команды изменения размера шрифта, поэтому его можно использовать совместно с обычным текстом внутри абзаца.

По умолчанию определены только латинские буквы, цифры и некоторые из стандартных символов. Для определения других символов можно воспользоваться макросом \DefineLCDchar. Макросу передаётся имя символа и битовая маска, определяющая картинку 5×7 точек. Имя символа может быть однобуквенным, тогда соответствующая буква замещается новым рисунком, или многобуквенным, тогда созданный рисунок кодируется указанным словом в фигурных скобках. Другие размеры матрицы в пакете отсутствуют, но при желании его вполне можно доработать.

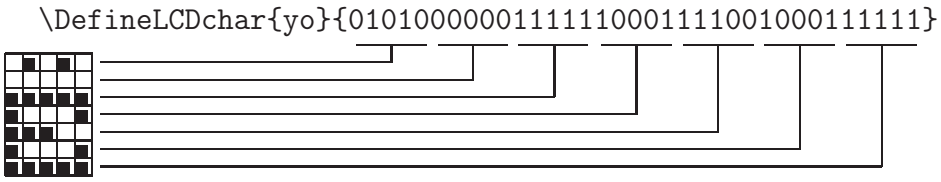
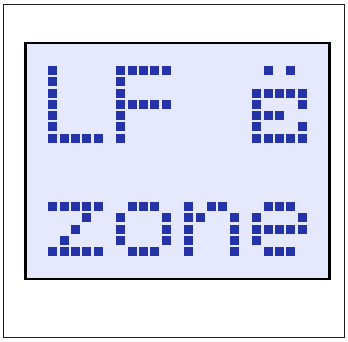


Рис. 5.2. Определяем букву «Ё» для LCD

Для эмуляции дисплея используется команда \LCD в качестве обязательных параметров ей передаётся число строк и число столбцов, за которыми следует содержание строк, разделённых каким-то разделителем. В приведённом пример в качестве разделителя используется вертикальная черта, но вместо неё может быть любой символ:

```
\DefineLCDchar{yo}{0101000000111111000111100100011111 1}
\definecolor{lightblue}{rgb}{0.9,0.91,0.99}
\definecolor{darkblue}{rgb}{0.14,0.2,0.66}
\LCDcolors{darkblue}{lightblue}
\LCDframe
\setlength{\LCDunitlength}{1.5mm}
\LCD{2}{4}|LF {yo} |
          |zone |
```



5.1.3. Битовые поля

Для описания сетевых протоколов, а так же для бинарных форматов данных удобнее всего представить последовательность битов графически, то есть в виде таблицы. Это специализация пакета **bytefield**. В пакете определено одноимённое окружение `bytefield` в качестве обязательного аргумента которому передаётся ширина таблицы в битах:

\begin{bytefield}{«битовая ширина поля»}
  «битовые поля»
\end{bytefield}

В окружении `bytefield` работают команды \wordbox и \bitbox, которые формируют поля занимающие ширину таблицы или только часть её, соответственно:

\wordbox[«рамка»]{«число строк»}{«текст»}
\bitbox[«рамка»]{«число занимаемых битов»}{«текст»}

Не обязательный параметр «рамка» позволяет сформировать обрамление для текущего битового поля. Значение по умолчанию `[lrtb]` означает, что рамка рисуется со всех сторон поля: `l` — слава, `r` — справа, `t` — сверху и `b` — снизу. Строки разделяются двойной обратной чертой \\.

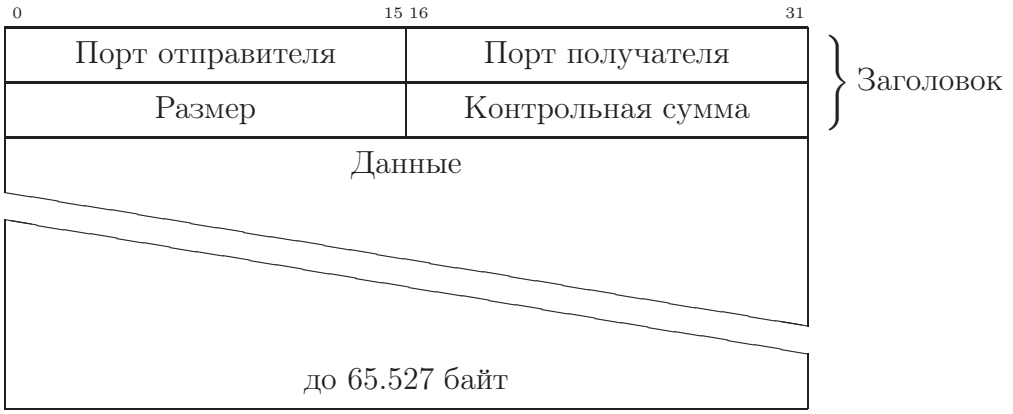


Таблица 5.1. Формат пакета UDP

Примерно следующим образом описывается формат пакета сетевого протокола UDP[1]:

```
\begin{bytefield}{32}
\bitheader{0,15,16,31}\\
\wordgroupr{Заголовок}
\bitbox{16}{Порт отправителя}\bitbox{16}{Порт получателя}\\
\bitbox{16}{Размер}\bitbox{16}{Контрольная сумма}
\endwordgroupr\\
\wordbox[lrt]{1}{Данные}\\
\skippedwords\\
\wordbox[lrb]{1}{до 65{.}527 байт}
\end{bytefield}
```

Кроме уже упомянутых команд создания полей при описании формата UDP использовалась команда нумерации столбцов \bitheader, конструкция для создания группы \wordgroupr и макрос \skippedwords для формирования «разрыва».

В качестве обязательного аргумента команде \bitheader передаётся список нумеруемых битов, при этом можно передавать диапазоны чисел, например, `{0-31}`. В пакете определены два окружения для группировки битовых полей \wordgroupr и \wordgroupl — отличие этих команд в том, что для первой заголовок группы вводится справа, а для второй — слева. Для более подробной информации следует обратиться к документации пакета.

5.2. Форматирование кода

LaTeX может использоваться не только для набора математики. Хотя набор математики безусловно вершина типографского искусства, но есть масса задач, где

[1] User Datagram Protocol — это сетевой протокол для передачи данных в сетях IP.

сложность форматирования сравнима. Представление исходников программ — это весьма не простое занятие, особенно если хочется сделать код читаемым.

5.2.1. verbatim

Самый простейший случай включение кода в текст — это заключить его в стандартное окружение `verbatim`, которое просто выводит этот текст на печать с отступами пользователя. Но при этом исчезает возможность делать акценты в избранных местах.

Небольшие вставки можно делать с помощью команды **verb**!текст!. Сразу после `\verb` должен идти группирующий символ (в данном случае «!»), который обязан указать окончание действия команды. Группирующий символ может быть любым, кроме пробела или звёздочки «*».

Пакет **verbatim** из коллекции **tools** переопределяет стандартную команду так, что внутри окружения можно вставлять тексты неограниченного размера. Кроме этого пакет предоставляет команду \verbatiminput, которой в качестве основного аргумента можно передать имя внешнего файл.

Кроме упомянутых окружения и макросов определены такие же, но со звёздочкой в конце имени, то есть окружение `verbatim*` и команды `\verb*` и \verbatiminput∗. *-форма отличается от базовой тем, что все пробелы визуализируются с помощью символа «␣».

```
\begin{verbatim*}
\textbf{verbatim} в         \LaTeX{}
\end{verbatim*}
```

```
\textbf{verbatim}␣в␣␣␣␣␣␣␣␣\LaTeX{}
```

Стандартный пакет **altt** — это почти то же самое, что и `verbatim`, но позволяет использовать команды LaTeX внутри окружения, правда шрифт в любом случае остаётся фиксированной ширины, как для печатной машинки.

```
\begin{alltt}
 \textbf{altt} в          \LaTeX{}
\end{alltt}
```

```
 altt в          LaTeX
```

Гораздо более разнообразные средства управления выводом неформатированного текста предоставляет пакет **fancyvrb**. За подробной информацией следует обратиться к документации пакета.

5.2.2. listings

Пакет **listings** специализируется на оформлении программного кода. К пакету прилагается подробнейшая документация[2]. С помощью команд пакета можно включить как небольшие кусочки кода, так и целые его сегменты, ну и, естественно, файлы.

[2]Следует поискать файл `listings.pdf`

Для загрузки пакета **listings** необходимо добавить в заголовок следующие инструкции:

Listing 5.1. Заголовок listings

```
\usepackage{listings}
% подгружаемые языки — подробнее в документации listings
\lstloadlanguages{[LaTeX]TeX, bash, MetaPost, Fortran, Perl, C++, make}
% включаем кириллицу и добавляем кое-какие опции
\lstset{language=[LaTeX]TeX, % выбираем язык по умолчанию
        extendedchars=true,  % включаем не латиницу
        escapechar=|,        % |«выпадаем» в LaTeX|
        frame=tb,            % рамка сверху и снизу
        commentstyle=\itshape, % шрифт для комментариев
        stringstyle=\bfseries} % шрифт для строк
```

Сразу после загрузки пакета рекомендуется «подгрузить» используемые в тексте языки программирования[3] с помощью макроса \lstloadlanguages. В квадратных скобочках перед названием языка можно указать желательный диалект.

Команда \lstset позволяет устанавливать значения по умолчанию. Некоторые из полезных умолчаний перечислены ниже:

- Для того чтобы можно было печатать кириллицу, например в комментариях, следует определить переменную `extendedchars=true`[4].

- Опция `escapechar` позволяет при наборе кода пользоваться услугами LaTeX напрямую. Всё, что находится между выбранными символами, обрабатывается средствами LaTeX. Естественно, если выбранный символ (в данном случае «|») используется в отображаемом языке, то могут возникнуть проблемы при компиляции. Для того чтобы обнулить `escapechar` достаточно ничего не писать за знаком равно при следующем переопределении.

- Инструкция `frame=<POSITION>` позволяет рисовать рамку вокруг сегмента кода. На вход принимаются буквы `t` — обрамление сверху, `b` — снизу, `l` и `r` —

[3] Текущая версия пакета 1.3c поддерживает следующие языки (в скобках указаны диалекты): ABAP, ACSL, Ada (83, 95), Algol (60, 68), Ant, Assembler (x86masm), Awk (gnu, POSIX), bash, Basic (Visual), C (ANSI, Handel, Objective, Sharp), C++ (ANSI, GNU, ISO, Visual), Caml (light, Objective), Clean, Cobol (1974, 1985, ibm) Comal 80, csh, Delphi, Eiffel, Elan, erlang, Euphoria, Fortran (77, 90, 95), GCL, Gnuplot, Haskell, HTML, IDL (empty, CORBA), inform, Java (empty, AspectJ), JVMIS, ksh, Lisp (empty, Auto), Logo, make (empty, gnu), Mathematica (1.0, 3.0), Matlab, Mercury, MetaPost, Miranda, Mizar, ML, Modula-2, MuPAD, NASTRAN, Oberon-2, OCL (decorative, OMG), Octave, Oz, Pascal (Borland6, Standard, XSC), Perl, PHP, PL/I, Plasm, POV, Prolog, Promela, Python, R, Reduce, Rexx, RSL, Ruby, S (empty, PLUS), SAS, Scilab, sh, SHELXL, Simula (67, CII, DEC, IBM), SQL, tcl (empty, tk), TeX (AlLaTeX, common, LaTeX, plain, primitive), VBScript, Verilog, VHDL (empty, AMS), VRML (97), XML, XSLT.

[4] Если это не сработает, то необходимо обновить пакет до последней версии или сменить дистрибутив LaTeX на более подходящий.

слева и справа, соответственно. В случае `frame=trbl` будет нарисована простейшая одинарная рамка. Опция `frame=` эквивалентен отказу от обрамления. Если вместо прописных букв указать заглавные `frame=TRBL`, то рамка будет двойная. В пакете есть возможность сделать рамки посложнее.

Все команды, определённые в пакете **listings**, начинаются с префикса `lst`. Команда для включение небольших кусочков кода \ lstinline !код! аналогична по действию команде **verb**!текст!.

Сегмент кода оформляется с помощью окружения `lstlisting`:

```
\begin{lstlisting}[language=Perl,
 caption={Включение сегмента кода}]
# проверка для перезаписи
if (open(CHECK,"<$file")) {
 $cmd=$term->
  readline("Overwrite (yes/NO): ");
 if (lc($cmd) ne "yes") {die;}
 close(CHECK);}

\end{lstlisting}
```

Listing 5.2. Включение сегмента кода

```
# проверка для перезаписи
if (open(CHECK,"<$file")) {
 $cmd=$term->
  readline("Overwrite (yes/NO): ");
 if (lc($cmd) ne "yes") {die;}
 close(CHECK);}
```

Необязательный параметр может принять опции, специфичные для оформления этого куска кода. Например, опция `language` позволяет установить язык программирования отличный от выбранного по умолчанию, а `caption` создаёт подпись к фрагменту кода.

Файлы можно включать с помощью команды \ lstinputlisting :

```
%установка значений по умолчанию
\lstset{numbers=left , language=MetaPost ,
        backgroundcolor=\color{yellow},
        frame=shadowbox, rulesepcolor=\color{black}}
%вставка файла
\lstinputlisting[firstline=16, lastline=24,
  emph={forsuffixes ,text ,bpath},emphstyle={\color{red}},
  emph={[2]fill ,unfill},emphstyle={[2]\bfseries\underbar},
]{intro.mp}
```

```
16 vardef drawshadowed(expr dx,dy)(text t) =
17   fixsize(t);
18   forsuffixes s=t:
19     fill bpath.s shifted (dx,dy);
20     unfill bpath.s;
21     drawboxed(s);
22 %    draw pic(s) withcolor red; %цвет текста
23   endfor;
24 enddef;
```

С помощью опций `firstline` и `secondline` можно указать строки, которые следует вывести. В зависимости от выбора языка форматирование существенно меняется. Инструкция `numbers=left` нумерует строки слева.

Для работы с цветами лучше загрузить уже упоминавшийся ранее пакет **color**. Цвета хороши для выделения каких-то ключевых слов и подложки, за которую отвечает опция `backgroundcolor`. Возможности для определения своих «словариков» предоставляет опции `emph=<список ключевых слов>`. В начале списка может идти его метка в квадратных скобках, таким образом можно поддерживать одновременно несколько списков. С помощью опции `emphstyle` можно определить способ выделения ключевых слов.

Обычно код располагается прямо по месту основного текста, так как обсуждение исходников можно не прерывать в самом коде, благо есть комментарии. Но при желании можно воспользоваться опцией `float`, чтобы из фрагмента кода получился полноценный «плавающий» объект.

Пакет с учётом диалектов поддерживает свыше сотни распространённых языков программирования и разметки. Так что, скорее всего, Вам не придётся определять свой язык с помощью инструкции \lstdefinelanguage. Но если очень хочется, то и это возможно.

5.3. Представление алгоритмов

Собственно говоря, именно то, ради чего Д.Э. Кнут и создал TEX. Поэтому пакеты для облегчения записи алгоритмов в LATEX были с самого его рождения. На текущий момент число даже стандартных пакетов, подпадающих под эту тематику, больше десятка. Здесь рассмотрена только малая часть из них.

5.3.1. algorithms

Пакет **algorithms** ориентирован на написание алгоритмов, а не на представление кода. Это позволяет отрешиться от форматирования и сосредоточиться на основной задаче. Пакет определяет окружение `algorithmic`. Для использования в преамбуле следует загрузить одноимённый стиль. Если необязательный аргумент определён, то осуществляется нумерация строк. Если аргумент равен 1, то нумеруются все строки, если 2 — то каждая вторая, а далее по индукции.

Команда \STATE определяет простое утверждение. Условный оператор представлен командами \IF{<условие>}, \ELSIF{<условие>}, \ELSE и \ENDIF. Циклы представлены операторами \FOR и \FORALL, которые закрываются командой \ENDFOR. Аналогично присутствуют пары \WHILE{<условие>} — \ENDWHILE, \REPEAT — \UNTILL{<условие>} и бесконечный цикл \LOOP — \ENDLOOP. Кроме уже перечисленных конструкций определены предварительное условие для корректного выполнение алгоритма \REQUIRE, постусловие, которое должно выполняться при корректной работе алгоритма, \ENSURE, возвращение результата \RETURN, промежуточная печать \PRINT и комментарий \COMMENT.

Algorithm 1 Пример использования пакета **algorithm**

```
\begin{algorithmic}[1]
\IF{\(i\leqslant0\)} \STATE \(i\gets1\) \ELSE
 \IF{\(i\geqslant0\)} \STATE \(i\gets0\)
 \COMMENT{смысла в этом алгоритме не ищите}
 \ENDIF
\ENDIF
\ENSURE \(i\geqslant0\)
\FORALL{\(\xi \in \mathcal{A}\)}
 \STATE \(\mathcal{B}\gets\xi^2\)
\ENDFOR
\RETURN \(\mathcal{B}\)
\end{algorithmic}
```

1: **if** $i \leqslant 0$ **then**
2: $i \leftarrow 1$
3: **else**
4: **if** $i \geqslant 0$ **then**
5: $i \leftarrow 0$ {смысла в этом алгоритме не ищите}
6: **end if**
7: **end if**
Ensure: $i \geqslant 0$
8: **for all** $\xi \in \mathcal{A}$ **do**
9: $\mathcal{B} \leftarrow \xi^2$
10: **end for**
11: **return** $\mathcal{B}$

Собственно говоря, всё. Псевдокод автоматически разбивается на строки и форматируется в соответствии с общепринятыми представлениям. Очевидно также, что навыки набора математики будут здесь очень кстати. Подробности по настройки пакета следует выяснять в документации к нему: `algorithms.pdf`.

Для того чтобы из объекта `algorithmic` сделать «плавающий объект» можно воспользоваться окружением `algorithm`, для использования следует в преамбуле загрузить одноимённый стиль. Внутри `algorithm` можно использовать команды **caption** и **label**.

5.3.2. Клоны algorithm

Используя имеющиеся наработки пакета **algorithm**, был создан **algorithmicx**. Этот пакет предоставляет более расширенный набор команд. Кроме этого пользователю предоставляются команды, которые позволяют сформировать свои алгоритмические конструкции. Автор так же предоставил вариант форматирования отступов принятый в Pascal, что позволяет относительно легко переводить программы на этом языке к виду, годному для красивой распечатки. Пакет использует те же окружения, что и **algorithm**, что приводит к несовместимости этих двух пакетов.

Решение схожей функциональности предоставляет пакет **algorithm2e**. Форматирование С-подобно. Предоставлен избыточный набор конструкций и возможность самому создавать новые структуры. Есть зачатки локализации. Пакет использует окружение `algorithm`, что не позволяет работать совместно с одноимённым пакетом **algorithm**.

5.3.3. clrscode

Пакет **clrscode** представляет возможность набирать псевдокод, как это делали авторы книги «Алгоритмы: построение и анализ» Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест и Клиффорд Штайн[5]. Для работы с пакетом необходимо загрузить одноимённый стиль. Прекрасный пример того, как можно адаптировать LaTeX для нужд создания книг по программированию.

```
\begin{codebox}
\Procname{
     $\proc{Сортировка методом вставок}$}
\li \For $j \gets 2$ \To $\id{length}[A]$
\li \Do $\id{key} \gets A[j]$
\li $i \gets j-1$
\li \While $i > 0$ and $A[i] > \id{key}$
\li \Do $A[i+1] \gets A[i]$
\li $i \gets i-1$ \End
\li $A[i+1] \gets \id{key}$ \End
\end{codebox}
```

Сортировка методом вставок

```
1  for j ← 2 to length[A]
2       do key ← A[j]
3          i ← j − 1
4          while i > 0 and A[i] > key
5               do A[i + 1] ← A[i]
6                  i ← i − 1
7          A[i + 1] ← key
```

Рис. 5.3. Пример использования пакета **clrscode**

5.3.4. pseudocode

Профессора Дональд Л. Крехер (Donald L. Kreher) и Дуглас Р. Стинсон (Douglas R. Stinson) написали книгу «Combinatorial Algorithms: Generation, Enumeration and Search». Специально для этой книги в целях написания псевдокода они создали пакет, который так и назвали: **pseudocode**. Дональд Л. Крехер использовал одноимённое окружение и в своей следующей книге по алгоритмам, выпущенной уже 2005 году. Пакет развивается и поддерживается.

5.4. Заключение

К сожалению в книгах по LaTeX редко рассматриваются структуры полезные для представления программных текстов или псевдокода. Здесь я попытался восполнить этот зияющий пробел. Тема настолько обширна, что разрабатывать её можно почти бесконечно. LaTeX сам по себе код, поэтому программистам, по идее, должно быть уютно в его окружении.

[5] *Introduction to algorithms*, Second Edition Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

```
\begin{pseudocode}{C2F\_таблица}
              {\text{от}, \text{до}}
\PROCEDURE{C2F}{c}
\COMMENT{Преобразование
   $^\circ$C$\to$$^\circ$F}\\
f \GETS {9c/5} + 32\\
\RETURN{f}
\ENDPROCEDURE
\MAIN
x \GETS \text{от} \\
\WHILE x \leqslant \text{до} \DO
\BEGIN
\OUTPUT{x, \CALL{C2F}{x}}\\
x \GETS x+1
\END
\ENDMAIN
\end{pseudocode}
```

Algorithm 1.1: C2F_таблица(от, до)

procedure C2F(c)
comment: Преобразование °C→°F
$f \leftarrow 9c/5 + 32$
return (f)

main
$x \leftarrow$ от
while $x \leqslant$ до
do { **output** $(x, \text{C2F}(x))$; $x \leftarrow x + 1$

Рис. 5.4. Пример использования пакета **pseudocode**

Исходники LaTeX и контроль версий

LaTeX-исходник тоже представляет из себя код. И как всякий код он достоин включение в систему контроля версий. Часто бывает любопытно узнать версию текущего документа и последний момент его обновления. Если в качестве системы контроля версий используется Subversion или svn, то для начала следует загрузить пакет **svn**[6].

```
\usepackage{svn}
\SVN $Date$
\SVN $Rev$
```

При этом в текст следует добавить метки, предваряемые командой \SVN. Для интерполяции меток в системе Subversion при обновлении файла следует выполнить команды вида:

```
> svn propset svn:keywords "Date Rev" «имя файла»
> svn commit -m "интерполяция меток"
```

При этом svn передаётся информация какие именно метки требуется обновлять при выполнении `commit`. В данном случае это метки `Date` и `Rev` — дата и версия, соответственно. Более подробную информацию можно получить с помощью команды

```
> svn help propset
```

[6]Если же в вашем проекте используется CVS (Concurrent Versions System), то следует воспользоваться пакетом **rcs**. За подробностями следует обратиться к документации пакета.

Команда \SVN $Date$ определяет команды \SVNDate и \SVNTime, ответственные за календарную дату и время. Все остальные команды вида \SVN $Keyword$, где `Keyword` — одна из интерполируемых меток svn, определяют команды \SVNKeyword.

После интерполяции метки будут выглядеть примерно следующим образом:

```
\SVN $Date: 2006-11-25 21:02:20 +0600 $
\SVN $Rev: 265 $
Документ обновлён \SVNDate\ \SVNTime

Текущая версия \SVNRev
```

Документ обновлён 25 ноября 2006 г. 21:02:20

Текущая версия 265

Схожую функциональность предоставляет пакет **svninfo**.