813. Разрыв абзацев на строки. [Breaking paragraphs into lines] Сейчас мы подходим, вероятно, к самому интересному алгоритму TEX'а — механизму выбора «наилучших возможных» точек разрыва, которые дают отдельные строки абзаца. Алгоритм разрыва строк TEX'а принимает данный горизонтальный список и преобразует его в последовательность рамок, которые затем присоединяются к текущему вертикальному списку. В ходе выполнения этого, он создаёт особую структуру данных, содержащую три вида записей, которые больше ни где TEX'ом не используются. Такие узлы создаются только во время обработки абзаца, а позже они удаляются из памяти; таким образом, другим частям TEX'а не нужно ни чего знать о том, как выполняется разрыв строк.

Используемый здесь способ основан на подходе, разработанном Michael F. Plass и автором в 1977, в последствии обобщённым и улучшенным теми же двумя людьми в 1980. Подробное обсуждение появилось в *SOFTWARE—Practice & Experience* **11** (1981), 1119–1184, где показано, что задача разрыва строк может рассматриваться как особый случай задачи вычисления кратчайшего пути в ациклической сети. Упомянутая статья включает множество примеров и описывает историю вопроса разбиения абзацев на строки, как оно производилось печатниками на протяжении столетий. Данная реализация добавила две новых идеи в алгоритм 1980-го года: требования к объёму памяти значительно уменьшились за счёт использования записей меньшего размера для неактивных узлов, чем для активных, а арифметическое переполнение предотвращается за счёт использования «разностных расстояний» ["delta distances"] вместо того, чтобы отслеживать всё расстояние от начала абзаца до текущей точки.

814. Процедура *line_break* должна вызываться только в горизонтальном режиме; она выходит из этого режима и помещает результат своей работы в текущий вертикальный список объемлющего вертикального режима (или внутреннего вертикального режима). Имеется один явный параметр: *final_widow_penalty* — величина дополнительного штрафа, который должен вставляться перед последней строкой абзаца.

Есть также несколько неявных параметров: горизонтальный список, который нужно разорвать, начинается на *link*(*head*) и является непустым. Значение переменной *prev_graf* на объемлющем семантическом уровне указывает, где абзац должен начинаться в последовательности номеров строк, если используются подвешенный отступ [hanging indentation] или команда `\parshape`; *prev_graf* равна нулю, если только этот абзац не продолжается после выключенной формулы. Другие неявные параметры, такие как *par_shape_ptr* и различные штрафы, используемые для переносов слов, и т. д., появляются в массиве *eqtb*.

После того, как процедура *line_break* отработает, она обновит текущий вертикальный список и значение переменной *prev_graf*. Более того, глобальная переменная *just_box* будет указывать на последнюю рамку, созданную процедурой *line_break* так, что ширину этой строки можно установить, когда необходимо решить, использовать ли *above_display_skip* или *above_display_short_skip* перед выключенной формулой.

⟨ Общие переменные 13 ⟩ +≡
just_box: *pointer*; { узел *hlist_node* для последней строки нового абзаца }

815. Т. к. *line_break* является довольно длинной процедурой — своего рода маленьким миром в себе — мы должны составлять её понемногу, чуть более внимательно, чем делали с более простыми процедурами TEX'а. Вот её общий набросок.

```
⟨ Объяви подпроцедуры для line_break 826 ⟩
procedure line_break(final_widow_penalty : integer);
  label done, done1, done2, done3, done4, done5, continue;
  var ⟨ Локальные переменные для разрывов строк 862 ⟩
  begin pack_begin_line ← mode_line;  { это для сообщений о переполненной или разреженной рамке }
  ⟨ Готовься разбить строку 816 ⟩;
  ⟨ Найди наилучшие точки разрыва 863 ⟩;
  ⟨ Разорви абзац в выбранных точках, выровняй полученные строки, чтобы исправить ширины и
      добавить их в текущий вертикальный список 876 ⟩;
  ⟨ Приведи в порядок память, удаляя узлы разрыва 865 ⟩;
  pack_begin_line ← 0;
  end;
```

816. Первая задача — это переместить список из *head* в *temp_head* и вступить в объемлющий семантический уровень. Мы также присоединяем клей `\parfillskip` к концу абзаца, удаляя пробел (или другой узел клея), если он там был, т. к. пробелы обычно предшествуют пустым строкам и выключенным формулам (знакам '`$$`'). Клею *par_fill_skip* предшествует бесконечный штраф, поэтому он никогда не будет рассматриваться, как возможная точка разрыва.

Этот код предполагает, что узлы *glue_node* и *penalty_node* занимают одно и то же число слов массива *mem*.

```
⟨ Готовься разбить строку 816 ⟩ ≡
  link(temp_head) ← link(head);
  if is_char_node(tail) then tail_append(new_penalty(inf_penalty))
  else if type(tail) ≠ glue_node then tail_append(new_penalty(inf_penalty))
    else begin type(tail) ← penalty_node; delete_glue_ref(glue_ptr(tail)); flush_node_list(leader_ptr(tail));
      penalty(tail) ← inf_penalty;
      end;
  link(tail) ← new_param_glue(par_fill_skip_code); init_cur_lang ← prev_graf mod '200000;
  init_l_hyf ← prev_graf div '20000000; init_r_hyf ← (prev_graf div '200000) mod '100; pop_nest;
```

См. также разделы 827, 834 и 848.

Этот код используется в разделе 815.

817. Когда ищутся оптимальные разрывы строк, TEX создаёт «узел разрыва» ["break node"] для каждого разрыва, который *допусти́м*, в смысле, что существует способ закончить строку в данной точке, не растягивая её или какую-либо другую строку больше данного значения допуска [tolerance]. Узел разрыва характеризуется тремя составляющими: положением разрыва (которое является указателем на *glue_node*, *math_node*, *penalty_node* или *disc_node*); порядковым номером строки, которая будет следовать за этой точкой разрыва; и классом годности строки, которая только что закончилась, т. е., одним из значений *tight_fit* (плотная), *decent_fit* (подходящая), *loose_fit* (свободная) или *very_loose_fit* (очень свободная).

```
define tight_fit = 3  { класс годности для строк, сжатых от 0,5 до 1,0 их сжимаемости }
define loose_fit = 1  { класс годности для строк, растянутых от 0,5 до 1,0 от их растяжимости }
define very_loose_fit = 0  { класс годности для строк, растянутых больше их растяжимости }
define decent_fit = 2  { класс годности для всех остальных строк }
```

818. В сущности, алгоритм определяет лучший возможный путь достигнуть каждого допустимого сочетания положения, строки и годности. Таким образом, он отвечает на вопросы вроде того, «Какой лучший способ разорвать начальную часть абзаца так, чтобы четвёртая строка была плотной строкой, заканчивающейся в таком-то и таком-то месте?» Однако, тот факт, что все строки должны быть одинаковой длины после определённой точки, позволяет рассматривать все достаточно большие номера строк эквивалентными случаю, когда параметр looseness равен нулю, и это позволяет алгоритму сократить затраты памяти и времени.

«Активный узел» [“active node”] и «пассивный узел» [“passive node”] создаются в массиве *mem* для каждой допустимой точки разрыва, которую нужно рассматривать. Активные узлы занимают три слова, а пассивные узлы — два слова. Нам необходимы активные узлы только для точек разрывов около того места в абзаце, которое сейчас рассматривается, поэтому занимаемая ими память освобождается через сравнительно небольшое время после их создания.

819. Активный узел для данной точки разрыва содержит шесть полей:

link указывает на следующий узел в списке активных узлов; последний активный узел имеет *link* = *last_active*.

break_node указывает на пассивный узел связанный с этой точкой разрыва.

line_number — номер строки, следующей за этой точкой разрыва.

fitness — класс годности строки, заканчивающейся на этой точке разрыва.

type принимает значение или *hyphenated*, или *unhyphenated*, в зависимости от того, является ли эта точка разрыва узлом *disc_node*.

total_demerits — минимально возможная сумма дефектностей всех строк, идущих от начала абзаца до этой точки разрыва.

Значение переменной *link*(*active*) указывает на первый активный узел в связном списке текущих активных узлов. Этот список упорядочен по номеру строки (*line_number*), за исключением того, что узлы с *line_number* > *easy_line* могут быть связаны друг с другом в любом порядке.

define *active_node_size* = 3 { число слов в активных узлах }
define *fitness* ≡ *subtype* { *very_loose_fit* . . *tight_fit* на последней строке для этого разрыва }
define *break_node* ≡ *rlink* { указатель на соответствующий пассивный узел }
define *line_number* ≡ *llink* { строка, которая начинается с этой точки разрыва }
define *total_demerits*(#) ≡ *mem*[# + 2].*int* { величина, которую TEX минимизирует }
define *unhyphenated* = 0 { поле *type* обычного активного узла разрыва }
define *hyphenated* = 1 { поле *type* активного узла, который разрывает на узле *disc_node* }
define *last_active* ≡ *active* { активный список заканчивается там же, где он и начинается }

820. ⟨ Настрой заголовки особого списка и узлы констант 790 ⟩ +≡
type(*last_active*) ← *hyphenated*; *line_number*(*last_active*) ← *max_halfword*; *subtype*(*last_active*) ← 0;
{ поле *subtype* никогда не проверяется этим алгоритмом }

821. Пассивный узел для данной точки разрыва содержит только четыре поля:

link указывает на пассивный узел, созданный сразу перед этим узлом, если таковой был, иначе поле содержит значение *null*.

cur_break указывает на положение этой точки разрыва в горизонтальном списке для разрываемого абзаца.

prev_break указывает на пассивный узел, который должен предшествовать данному узлу в оптимальном пути к данной точке разрыва.

serial равно n, если этот пассивный узел является n-ым узлом, созданным во время текущего прохода. (Это поле используется только, когда распечатывается детальная статистика о расчётах разрывов строк.)

Существует глобальная переменная, называемая *passive*, которая указывает на последний созданный пассивный узел. Другая глобальная переменная, *printed_node*, используется, чтобы помочь распечатать абзац, когда подробные данные о вычислениях точек разрывов отображаются на экране.

```
define passive_node_size = 2  { число слов в пассивных узлах }
define cur_break ≡ rlink  { в пассивном узле, указывает на положение этой точки разрыва }
define prev_break ≡ llink
        { указывает на пассивный узел, который должен предшествовать этому узлу }
define serial ≡ info  { порядковый номер для символьного определения }
```

```
⟨ Общие переменные 13 ⟩ +≡
passive: pointer;  { самый последний узел в пассивном списке }
printed_node: pointer;  { самый последний узел, который распечатали }
pass_number: halfword;  { число пассивных узлов, размещённых в этом проходе }
```

822. Активный список также содержит узлы «delta», которые помогают алгоритму вычислять негодность отдельных строк. Такие узлы появляются только между двумя активными узлами и они имеют поле $type = delta\_node$. Если p и r активные узлы, и если q является узлом delta между ними, так, чтобы $link(p) = q$ и $link(q) = r$, то q хранит разность расстояний между строками в горизонтальном списке, которые начинаются после точки разрыва p, и строками, которые начинаются после точки разрыва r. Другими словами, если мы знаем длину строки, которая начинается после p и заканчивается в нашей текущей позиции, то соответствующую длину строки, которая начинается после r, можно получить, добавив величину из узла q. Узел delta содержит шесть масштабных чисел, т. к. он должен хранить общее изменение растяжимости клея по отношению ко всем порядкам бесконечности. Разность естественной ширины появляется в $mem[q+1].sc$; разности растяжимостей в единицах pt, fil, fill и filll появляются в $mem[q+2\,..\,q+5].sc$; и разность сжимаемости появляется в $mem[q+6].sc$. Поле *subtype* узла delta не используется.

```
define delta_node_size = 7  { число слов в узле delta }
define delta_node = 2  { поле type в узле delta }
```

823. Когда алгоритм работает, он обслуживает набор из шести delta-подобных регистров для длины строки, следующей за первой активной точкой разрыва, до текущего положения в данном горизонтальном списке. Когда он проходит по активному списку, он также обслуживает подобный набор из шести регистров для длины, следующей рассматриваемой активной точки разрыва. Третий набор хранит длину пустой строки (а именно, сумму `\leftskip` и `\rightskip`); а четвёртый набор используется, чтобы создавать новые delta-узлы.

Когда мы проходим узел delta, мы хотим выполнять действия, подобные

$$\textbf{for } k \leftarrow 1 \textbf{ to } 6 \textbf{ do } \mathit{cur\_active\_width}[k] \leftarrow \mathit{cur\_active\_width}[k] + \mathit{mem}[q+k].\mathit{sc},$$

но мы хотим выполнять их без циклов **for** . Макрос *do_all_six* делает такое присвоение значений шести элементам удобным.

define *do_all_six*(#) ≡ #(1); #(2); #(3); #(4); #(5); #(6)

⟨ Общие переменные 13 ⟩ +≡
active_width: **array** [1 .. 6] **of** *scaled*; { расстояние от первого активного узла до *cur_p* }
cur_active_width: **array** [1 .. 6] **of** *scaled*; { расстояние от текущего активного узла }
background: **array** [1 .. 6] **of** *scaled*; { длина «пустой» строки }
break_width: **array** [1 .. 6] **of** *scaled*; { длина, вычисляемая после текущего разрыва }

824. Давайте, сформулируем принципы узлов delta более точно и кратко, так, чтобы следующие программы были более понятными. Для каждой допустимой точки разрыва p в абзаце, мы определяем две величины $\alpha(p)$ и $\beta(p)$ такие, что длина материала в строке от точки разрыва p до точки разрыва q есть $\gamma+\beta(q)-\alpha(p)$, для некоторого фиксированного γ. Интуитивно, $\alpha(p)$ и $\beta(q)$ являются общей длиной данных от начала абзаца до точки «после» разрыва на p и к точке «перед» разрывом на q, а γ является шириной пустой строки, а именно длиной, добавляемой параметрами `\leftskip` и `\rightskip`.

Предположим, например, что абзац полностью состоит из чередующихся рамок и промежутков клея; пусть рамки имеют ширины $x_1 \ldots x_n$, и пусть пропуски имеют ширины $y_1 \ldots y_n$, так что абзац может быть представлен формулой $x_1 y_1 \ldots x_n y_n$. Пусть p_i будет допустимой точкой разрыва на y_i; тогда $\alpha(p_i) = x_1 + y_1 + \cdots + x_i + y_i$ и $\beta(p_i) = x_1 + y_1 + \cdots + x_i$. Чтобы проверить это, заметим, что длина материала от p_2 до p_5, скажем, есть $\gamma + x_3 + y_3 + x_4 + y_4 + x_5 = \gamma + \beta(p_5) - \alpha(p_2)$.

Величины α, β, γ включают растяжимость и сжимаемость клея, а также естественную ширину. Если мы должны были бы вычислить $\alpha(p)$ и $\beta(p)$ для каждого p, нам нужно было бы выполнять много точных арифметических вычислений, и очень точные числа должны были бы храниться в активных узлах. TEX избегает этой проблемы, работая полностью с относительными разностями или с «дельтами» (“deltas”). Предположим, например, что активный список содержит $a_1\,\delta_1\,a_2\,\delta_2\,a_3$, где a-ые [элементы] являются активными точками разрыва, а δ-ые [элементы] являются узлами delta. Тогда $\delta_1 = \alpha(a_1) - \alpha(a_2)$ и $\delta_2 = \alpha(a_2) - \alpha(a_3)$. Если алгоритм разрыва строк сейчас остановился на некоторой другой точке разрыва p, массив *active_width* содержит значение $\gamma + \beta(p) - \alpha(a_1)$. Если мы сканируем список активных узлов и рассматриваем предварительную строку, которая проходит от a_2 до p, скажем, массив *cur_active_width* будет содержать значение $\gamma + \beta(p) - \alpha(a_2)$. Таким образом, когда мы перемещаемся от a_2 до a_3, мы хотим добавить $\alpha(a_2) - \alpha(a_3)$ к *cur_active_width*; и это есть именно δ_2, которое появилось в активном списке между a_2 и a_3. Массив *background* содержит γ. Массив *break_width* будет использоваться, чтобы вычислять значения новых узлов delta, когда активный список обновляется.

825. Узлы клея в горизонтальном списке, который преобразуется в абзац, не должны включать «бесконечную» сжимаемость; поэтому алгоритм содержит четыре регистра для растяжения и только один для сжатия. Если пользователь попытается ввести бесконечную сжимаемость, сжимаемость будет вновь установлена конечной, и будет выведено сообщение об ошибке. Булева переменная *no_shrink_error_yet* предотвращает появление этого сообщения об ошибке более одного раза в одном абзаце.

```
define check_shrinkage(#) ≡
        if (shrink_order(#) ≠ normal) ∧ (shrink(#) ≠ 0) then
          begin # ← finite_shrink(#);
          end
```

⟨ Общие переменные 13 ⟩ +≡
```
no_shrink_error_yet: boolean;   { мы уже сообщали о бесконечной сжимаемости? }
```

826. ⟨ Объяви подпроцедуры для *line_break* 826 ⟩ ≡
```
function finite_shrink(p : pointer): pointer;   { избавляет от бесконечной сжимаемости }
  var q: pointer;   { новая спецификация клея }
  begin if no_shrink_error_yet then
    begin no_shrink_error_yet ← false; print_err("Infinite␣glue␣shrinkage␣found␣in␣a␣paragraph");
    help5("The␣paragraph␣just␣ended␣includes␣some␣glue␣that␣has")
    ("infinite␣shrinkability,␣e.g.,␣`\hskip␣0pt␣minus␣1fil´.")
    ("Such␣glue␣doesn´t␣belong␣there---it␣allows␣a␣paragraph")
    ("of␣any␣length␣to␣fit␣on␣one␣line.␣But␣it´s␣safe␣to␣proceed,")
    ("since␣the␣offensive␣shrinkability␣has␣been␣made␣finite."); error;
    end;
  q ← new_spec(p); shrink_order(q) ← normal; delete_glue_ref(p); finite_shrink ← q;
  end;
```

См. также разделы 829, 877, 895 и 942.

Этот код используется в разделе 815.

827. ⟨ Готовься разбить строку 816 ⟩ +≡
```
no_shrink_error_yet ← true;
check_shrinkage(left_skip); check_shrinkage(right_skip);
q ← left_skip; r ← right_skip; background[1] ← width(q) + width(r);
background[2] ← 0; background[3] ← 0; background[4] ← 0; background[5] ← 0;
background[2 + stretch_order(q)] ← stretch(q);
background[2 + stretch_order(r)] ← background[2 + stretch_order(r)] + stretch(r);
background[6] ← shrink(q) + shrink(r);
```

828. Переменная указателя *cur_p* проходит через данный горизонтальный список, когда мы ищем точки разрывов. Эта переменная глобальная, т. к. она используется как в процедуре *line_break*, так и в её подпрограмме *try_break*.

Другая глобальная переменная, называемая *threshold*, используется, чтобы определить допустимость отдельных строк: точки разрывов допустимы, если существует способ достигнуть их, не создавая строк, негодность которых превысит порог *threshold*. (Негодность сравнивается с величиной *threshold* перед тем, как штрафы добавляются для того, чтобы значения штрафов не влияли на допустимость точек разрывов кроме того, что разрыв не допускается, когда штраф равен 10000 или больше.) Если порог *threshold* равен 10000 или больше, все возможные разрывы считаются допустимыми, т. к. функция *badness*, определённая выше, никогда не возвращает значение больше 10000.

Вплоть до трёх проходов может быть выполнено через абзац в попытке найти по крайней мере один набор допустимых точек разрыва. На первом проходе, мы имеем *threshold* = *pretolerance* и *second_pass* = *final_pass* = *false*. Если в результате этого прохода не нашлось допустимого решения, порог *threshold* устанавливается равным *tolerance*, *second_pass* устанавливается равным *true* и делается попытка перенести как можно больше слов. Если она также неудачна, мы добавляем *emergency_stretch* к растяжимости пустой строки background и устанавливаем *final_pass* = *true*.

⟨ Общие переменные 13 ⟩ +≡
cur_p: *pointer*; { рассматриваемая сейчас точка разрыва }
second_pass: *boolean*; { эта наша вторая попытка разбить абзац на строки? }
final_pass: *boolean*; { эта наша последняя попытка разбить этот абзац? }
threshold: *integer*; { наибольшая негодность допустимых строк }

829. Сердце процедуры разбивки абзаца на строки — подпрограмма '*try_break*', которая проверяет, является ли текущая точка разрыва *cur_p* допустимой, просматривая активный список, чтобы увидеть, какие строки текста могут быть сделаны из активных узлов до *cur_p*. Если допустимые разрывы возможны, создаются новые узлы разрывов. Если *cur_p* находится слишком далеко от активного узла, этот узел деактивируется.

Параметр *pi* для процедуры *try_break* является штрафом, связанным с разрывом на узле *cur_p*; мы имеем $pi = eject\_penalty$, если разрыв является принудительным, и $pi = inf\_penalty$, если разрыв недопустим.

Другой параметр, *break_type*, устанавливается *hyphenated* или *unhyphenated*, в зависимости от того, находится ли текущий разрыв на *disc_node* или нет. Конец абзаца также рассматривается как '*hyphenated*'; этот случай различается по условию $cur\_p = null$.

```
define copy_to_cur_active(#) ≡ cur_active_width[#] ← active_width[#]
define deactivate = 60  { Идти сюда, когда узел r следует деактивировать }
⟨ Объяви подпроцедуры для line_break 826 ⟩ +≡
procedure try_break(pi : integer; break_type : small_number);
  label exit, done, done1, continue, deactivate;
  var r: pointer;  { проходит по активному списку }
    prev_r: pointer;  { отстаёт на шаг от r }
    old_l: halfword;  { наибольший номер строки в текущем классе эквивалентности строк }
    no_break_yet: boolean;  { мы нашли допустимый разрыв на узле cur_p? }
    ⟨ Другие локальные переменные для try_break 830 ⟩
  begin ⟨ Убедись, что pi в верных пределах 831 ⟩;
  no_break_yet ← true; prev_r ← active; old_l ← 0; do_all_six(copy_to_cur_active);
  loop begin continue: r ← link(prev_r); ⟨ Если узел r типа delta_node, обнови cur_active_width,
      установи prev_r и prev_prev_r, затем goto continue 832 ⟩;
    ⟨ Если класс номеров строк завершился, создай новые активные узлы для наилучших годных
      разрывов в этом классе; затем return если r = last_active, иначе вычисли новую line_width 835
      ⟩;
    ⟨ Рассмотри дефектность строки от r до cur_p; деактивируй узел r, если он не должен больше
      быть активным, затем goto continue, если строка от r до cur_p негодная, иначе запиши
      новый годный разрыв 851 ⟩;
    end;
exit: stat ⟨ Обнови значение printed_node для символьного отображения 858 ⟩ tats
  end;
```

830. ⟨ Другие локальные переменные для *try_break* 830 ⟩ ≡
prev_prev_r: *pointer*; { отстаёт на шаг от *prev_r*, если *type*(*prev_r*) = *delta_node* }
s: *pointer*; { пробегает узлы перед [узлом] *cur_p* }
q: *pointer*; { указывает на новый создаваемый узел }
v: *pointer*; { указывает на спецификацию клея или узел впереди (ahead of) *cur_p* }
t: *integer*; { счётчик узлов, если *cur_p* является узлом переноса по усмотрению }
f: *internal_font_number*; { используется в расчёте ширины символа }
l: *halfword*; { номер строки текущего активного узла }
node_r_stays_active: *boolean*; { должен ли узел *r* оставаться в активном списке? }
line_width: *scaled*; { Текущая строка будет подгоняться к этой ширине }
fit_class: *very_loose_fit* .. *tight_fit*; { возможный класс годности тестируемой строки }
b: *halfword*; { негодность тестируемой строки }
d: *integer*; { дефектность тестируемой строки }
artificial_demerits: *boolean*; { был ли *d* принудительно установлен в ноль? }
save_link: *pointer*; { временно хранит значение поля *link*(*cur_p*) }
shortfall: *scaled*; { используется в расчётах негодности }

Этот код используется в разделе 829.

831. ⟨ Убедись, что *pi* в верных пределах 831 ⟩ ≡
if *abs*(*pi*) ⩾ *inf_penalty* **then**
if *pi* > 0 **then return** { эта точка разрыва не допускается бесконечным штрафом }
else *pi* ← *eject_penalty* { эта точка разрыва будет принудительной }

Этот код используется в разделе 829.

832. Следующий код использует то, что *type*(*last_active*) ≠ *delta_node*.
define *update_width*(#) ≡ *cur_active_width*[#] ← *cur_active_width*[#] + *mem*[*r* + #].*sc*
⟨ Если узел *r* типа *delta_node*, обнови *cur_active_width*, установи *prev_r* и *prev_prev_r*, затем **goto** *continue* 832 ⟩ ≡
if *type*(*r*) = *delta_node* **then**
begin *do_all_six*(*update_width*); *prev_prev_r* ← *prev_r*; *prev_r* ← *r*; **goto** *continue*;
end

Этот код используется в разделе 829.

833. Когда мы рассматриваем различные способы закончить строку на узле *cur_p*, в данном классе номеров строк, мы следим за лучшими общими известными дефектностями, в массиве с одним элементом для каждого из классов годности. Например, *minimal_demerits*[*tight_fit*] содержит наименьшие общие дефектности допустимых разрывов строк, заканчивающихся на узле *cur_p* с *tight_fit* строкой; *best_place*[*tight_fit*] указывает на пассивный узел для разрыва перед *cur_p*, который достигает такого оптимума; а *best_pl_line*[*tight_fit*] является полем *line_number* в активном узле, соответствующем *best_place*[*tight_fit*]. Когда нет известной допустимой последовательности разрывов, элементы массива *minimal_demerits* будут равны величине *awful_bad*, которая равна $2^{30} - 1$. Другая переменная, *minimum_demerits*, следит за наименьшим значением в массиве *minimal_demerits*.

define *awful_bad* ≡ '7777777777 { дефектность более миллиарда }

⟨ Общие переменные 13 ⟩ +≡
minimal_demerits: **array** [*very_loose_fit* .. *tight_fit*] **of** *integer*; { лучшая общая дефектность данной годности, известная для текущего класса строк и положения }
minimum_demerits: *integer*;
{ лучшая общая дефектность, известная для текущего класса строк и положения }
best_place: **array** [*very_loose_fit* .. *tight_fit*] **of** *pointer*; { как достигнуть *minimal_demerits* }
best_pl_line: **array** [*very_loose_fit* .. *tight_fit*] **of** *halfword*; { соответствующий номер строки }

834. ⟨ Готовься разбить строку 816 ⟩ +≡
minimum_demerits ← *awful_bad*; *minimal_demerits*[*tight_fit*] ← *awful_bad*;
minimal_demerits[*decent_fit*] ← *awful_bad*; *minimal_demerits*[*loose_fit*] ← *awful_bad*;
minimal_demerits[*very_loose_fit*] ← *awful_bad*;

835. Первая часть следующего кода является частью внутреннего цикла TEX'а, поэтому мы не хотим любых напрасных трат времени. Текущий активный узел, а именно узел r, содержит номер строки, который будет рассматриваться следующим. В конце списка мы разместили структуру данных так, чтобы $r = last\_active$ и $line\_number(last\_active) > old\_l$.

⟨ Если класс номеров строк завершился, создай новые активные узлы для наилучших годных разрывов в этом классе; затем **return** если $r = last\_active$, иначе вычисли новую *line_width* 835 ⟩ ≡

```
begin l ← line_number(r);
if l > old_l then
  begin  { сейчас мы более не находимся во внутреннем цикле }
  if (minimum_demerits < awful_bad) ∧ ((old_l ≠ easy_line) ∨ (r = last_active)) then
    ⟨ Создай новые активные узлы для лучших найденных годных разрывов 836 ⟩;
  if r = last_active then return;
  ⟨ Рассчитай новую ширину строки 850 ⟩;
  end;
end
```

Этот код используется в разделе 829.

836. Не нужно создавать новые активные узлы с *minimal_demerits* больше, чем *minimum_demerits* + *abs*(*adj_demerits*), т. к. такие активные узлы никогда не будут выбираться в окончательном разрыве абзацев. Это замечание позволяет нам оставить значительное число возможных точек разрыва без дальнейшего рассмотрения.

⟨ Создай новые активные узлы для лучших найденных годных разрывов 836 ⟩ ≡

```
begin if no_break_yet then ⟨ Вычисли значения break_width 837 ⟩;
⟨ Вставь узел delta, чтобы подготовиться к разрыву на cur_p 843 ⟩;
if abs(adj_demerits) ⩾ awful_bad − minimum_demerits then minimum_demerits ← awful_bad − 1
else minimum_demerits ← minimum_demerits + abs(adj_demerits);
for fit_class ← very_loose_fit to tight_fit do
  begin if minimal_demerits[fit_class] ⩽ minimum_demerits then
    ⟨ Вставь новый активный узел от best_place[fit_class] до cur_p 845 ⟩;
  minimal_demerits[fit_class] ← awful_bad;
  end;
minimum_demerits ← awful_bad;
⟨ Вставь узел delta, чтобы подготовить следующий активный узел 844 ⟩;
end
```

Этот код используется в разделе 835.

837. Когда мы вставляем новый активный узел для разрыва на узле *cur_p*, мы предполагаем, что новый узел должен помещаться сразу перед активным узлом *a*; затем по сути мы хотим вставить 'δ *cur_p* δ'' перед *a*, где $\delta = \alpha(a) - \alpha(cur\_p)$ и $\delta' = \alpha(cur\_p) - \alpha(a)$ в объяснённой выше нотации. Массив *cur_active_width* теперь хранит $\gamma + \beta(cur\_p) - \alpha(a)$; поэтому δ может быть получено вычитанием *cur_active_width* из величины $\gamma + \beta(cur\_p) - \alpha(cur\_p)$. Последняя величина может рассматриваться как длина строки «от *cur_p* до *cur_p*»; мы называем её *break_width* на *cur_p*.

Значение *break_width* обычно отрицательно, т. к. состоит из длины пустой строки background (которая обычно равна нулю) за вычетом ширины узлов, следующих за *cur_p*, которые удаляются после разрыва. Если, например, узел *cur_p* является узлом клея, ширина этого клея вычитается из background; и мы готовимся удалить все последующие узлы клея, штрафов, кернов и математические узлы, вычитая также их ширины.

Узлы кернов не исчезают на разрыве строки, если только они не являются явными (*explicit*).

define *set_break_width_to_background*(#) ≡ *break_width*[#] ← *background*[#]

⟨ Вычисли значения *break_width* 837 ⟩ ≡
```
begin no_break_yet ← false; do_all_six(set_break_width_to_background); s ← cur_p;
if break_type > unhyphenated then
  if cur_p ≠ null then ⟨ Вычисли значения переноса по усмотрению break_width 840 ⟩;
while s ≠ null do
  begin if is_char_node(s) then goto done;
  case type(s) of
  glue_node: ⟨ Вычти клей из break_width 838 ⟩;
  penalty_node: do_nothing;
  math_node: break_width[1] ← break_width[1] − width(s);
  kern_node: if subtype(s) ≠ explicit then goto done
    else break_width[1] ← break_width[1] − width(s);
  othercases goto done
  endcases;
  s ← link(s);
  end;
done: end
```
Этот код используется в разделе 836.

838. ⟨ Вычти клей из *break_width* 838 ⟩ ≡
```
begin v ← glue_ptr(s); break_width[1] ← break_width[1] − width(v);
break_width[2 + stretch_order(v)] ← break_width[2 + stretch_order(v)] − stretch(v);
break_width[6] ← break_width[6] − shrink(v);
end
```
Этот код используется в разделе 837.

839. Когда *cur_p* суть разрыв по усмотрению, длина строки «от *cur_p* до *cur_p*» должна определяться верно, чтобы другие вычисления правильно выполнялись. Предположим, что текст перед разрывом на *cur_p* имеет длину l_0, текст после разрыва имеет длину l_1, а замещаемый текст имеет длину l. Предположим также, что q — узел, следующий за замещаемым текстом. Тогда длина строки от *cur_p* до q будет вычисляться как $\gamma + \beta(q) - \alpha(cur\_p)$, где $\beta(q) = \beta(cur\_p) - l_0 + l$. Действительная длина будет равна длине пустой строки background плюс l_1, поэтому длина от *cur_p* до *cur_p* должна быть $\gamma + l_0 + l_1 - l$. Если текст-после-разрыва в узле переноса по усмотрению пуст, разрыв также может отбросить q; в этом необычном случае мы вычитаем длину q и любые другие узлы, которые будут отброшены после разрыва по усмотрению.

Значение длины l_0 не нужно вычислять, т. к. процедура *line_break* поместит его в глобальную переменную *disc_width* перед вызовом *try_break*.

```
⟨ Общие переменные 13 ⟩ +≡
disc_width: scaled;   { длина материала переноса по усмотрению предшествующего разрыву }
```

840.
```
⟨ Вычисли значения переноса по усмотрению break_width 840 ⟩ ≡
  begin t ← replace_count(cur_p); v ← cur_p; s ← post_break(cur_p);
  while t > 0 do
    begin decr(t); v ← link(v); ⟨ Вычти ширину узла v из break_width 841 ⟩;
    end;
  while s ≠ null do
    begin ⟨ Добавь ширину узла s к break_width 842 ⟩;
    s ← link(s);
    end;
  break_width[1] ← break_width[1] + disc_width;
  if post_break(cur_p) = null then s ← link(v);   { узлы могут быть отбрасываемыми после разрыва }
  end
```

Этот код используется в разделе 837.

841. Замещаемые тексты и тексты разрыва по усмотрению должны содержать только узлы символов, узлы кернов, узлы лигатур и узлы рамок или линеек.

```
⟨ Вычти ширину узла v из break_width 841 ⟩ ≡
  if is_char_node(v) then
    begin f ← font(v); break_width[1] ← break_width[1] − char_width(f)(char_info(f)(character(v)));
    end
  else case type(v) of
    ligature_node: begin f ← font(lig_char(v));
      break_width[1] ← break_width[1] − char_width(f)(char_info(f)(character(lig_char(v))));
      end;
    hlist_node, vlist_node, rule_node, kern_node: break_width[1] ← break_width[1] − width(v);
    othercases confusion("disc1")
    endcases
```

Этот код используется в разделе 840.

842. ⟨ Добавь ширину узла s к *break_width* 842 ⟩ ≡

```
if is_char_node(s) then
  begin f ← font(s); break_width[1] ← break_width[1] + char_width(f)(char_info(f)(character(s)));
  end
else case type(s) of
  ligature_node: begin f ← font(lig_char(s));
    break_width[1] ← break_width[1] + char_width(f)(char_info(f)(character(lig_char(s))));
    end;
  hlist_node, vlist_node, rule_node, kern_node: break_width[1] ← break_width[1] + width(s);
  othercases confusion("disc2")
  endcases
```

Этот код используется в разделе 840.

843. Мы используем то обстоятельство, что $type(active) \neq delta\_node$.

```
define convert_to_break_width(#) ≡ mem[prev_r + #].sc ←
            mem[prev_r + #].sc − cur_active_width[#] + break_width[#]
define store_break_width(#) ≡ active_width[#] ← break_width[#]
define new_delta_to_break_width(#) ≡ mem[q + #].sc ← break_width[#] − cur_active_width[#]
```

⟨ Вставь узел delta, чтобы подготовиться к разрыву на *cur_p* 843 ⟩ ≡

```
if type(prev_r) = delta_node then { изменить существующий узел delta }
  begin do_all_six(convert_to_break_width);
  end
else if prev_r = active then { вначале delta узел не нужен }
    begin do_all_six(store_break_width);
    end
  else begin q ← get_node(delta_node_size); link(q) ← r; type(q) ← delta_node;
    subtype(q) ← 0; { поле subtype не используется }
    do_all_six(new_delta_to_break_width); link(prev_r) ← q; prev_prev_r ← prev_r; prev_r ← q;
    end
```

Этот код используется в разделе 836.

844. Когда следующий код выполняется, у нас уже вставлен по крайней мере один активный узел перед r, поэтому $type(prev\_r) \neq delta\_node$.

```
define new_delta_from_break_width(#) ≡ mem[q + #].sc ← cur_active_width[#] − break_width[#]
```

⟨ Вставь узел delta, чтобы подготовить следующий активный узел 844 ⟩ ≡

```
if r ≠ last_active then
  begin q ← get_node(delta_node_size); link(q) ← r; type(q) ← delta_node;
  subtype(q) ← 0; { поле subtype не используется }
  do_all_six(new_delta_from_break_width); link(prev_r) ← q; prev_prev_r ← prev_r; prev_r ← q;
  end
```

Этот код используется в разделе 836.

845. Когда мы создаём активный узел, мы также создаём соответствующий пассивный узел.

⟨ Вставь новый активный узел от *best_place*[*fit_class*] до *cur_p* 845 ⟩ ≡
```
begin q ← get_node(passive_node_size); link(q) ← passive; passive ← q; cur_break(q) ← cur_p;
stat incr(pass_number); serial(q) ← pass_number; tats
prev_break(q) ← best_place[fit_class];
q ← get_node(active_node_size); break_node(q) ← passive; line_number(q) ← best_pl_line[fit_class] + 1;
fitness(q) ← fit_class; type(q) ← break_type; total_demerits(q) ← minimal_demerits[fit_class];
link(q) ← r; link(prev_r) ← q; prev_r ← q;
stat if tracing_paragraphs > 0 then ⟨ Печатай символьное описание нового узла разрыва 846 ⟩;
tats
end
```
Этот код используется в разделе 836.

846. ⟨ Печатай символьное описание нового узла разрыва 846 ⟩ ≡
```
begin print_nl("@@"); print_int(serial(passive)); print(":␣line␣"); print_int(line_number(q) − 1);
print_char("."); print_int(fit_class);
if break_type = hyphenated then print_char("-");
print("␣t="); print_int(total_demerits(q)); print("␣->␣@@");
if prev_break(passive) = null then print_char("0")
else print_int(serial(prev_break(passive)));
end
```
Этот код используется в разделе 845.

847. Длина строк зависит от того, указал ли пользователь команду `\parshape` или `\hangindent`. Если указатель *par_shape_ptr* не пустой, то он указывает на $(2n+1)$-словную запись в *mem*, где поле *info* в первом слове содержит значение переменной n, а другие $2n$ слов содержат левые поля и длины строк для первых n строк абзаца; спецификация для строки n применяется ко всем последующим строкам. Если *par_shape_ptr* = *null*, очертание абзаца зависит от значения указателя n = *hang_after*. Если $n \geqslant 0$, подвешенный отступ применяется к строкам $n+1$, $n+2$, ..., иначе он применяется к строкам 1, ..., $|n|$. Когда подвешенный отступ действует, левое поле устанавливается равным *hang_indent*, если *hang_indent* $\geqslant 0$, иначе оно устанавливается равным 0; длина строки равна *hsize* − |*hang_indent*|. Обычно эти переменные установлены в значения *par_shape_ptr* = *null*, *hang_after* = 1 и *hang_indent* = 0. Заметим, что если *hang_indent* = 0, значение переменной *hang_after* неуместно.

⟨ Общие переменные 13 ⟩ +≡
```
easy_line: halfword; { номера строк > easy_line эквивалентны в узлах разрывов }
last_special_line: halfword; { номера строк > last_special_line все имеют ту же самую ширину }
first_width: scaled; { ширина всех строк ⩽ last_special_line, если не было указано \parshape }
second_width: scaled; { ширина всех строк > last_special_line }
first_indent: scaled; { левое поле, соответствующее first_width }
second_indent: scaled; { левое поле, соответствующее second_width }
```

848. Мы вычисляем значения *easy_line* и других локальных переменных, имеющих отношение к длине строки, когда процедура *line_break* выполняет начальную установку своих переменных.

```
⟨ Готовься разбить строку 816 ⟩ +≡
  if par_shape_ptr = null then
    if hang_indent = 0 then
      begin last_special_line ← 0; second_width ← hsize; second_indent ← 0;
      end
    else ⟨ Установи параметры длины строки, готовясь к подвешенному отступу 849 ⟩
  else begin last_special_line ← info(par_shape_ptr) − 1;
    second_width ← mem[par_shape_ptr + 2 ∗ (last_special_line + 1)].sc;
    second_indent ← mem[par_shape_ptr + 2 ∗ last_special_line + 1].sc;
    end;
  if looseness = 0 then easy_line ← last_special_line
  else easy_line ← max_halfword
```

849.

```
⟨ Установи параметры длины строки, готовясь к подвешенному отступу 849 ⟩ ≡
  begin last_special_line ← abs(hang_after);
  if hang_after < 0 then
    begin first_width ← hsize − abs(hang_indent);
    if hang_indent ⩾ 0 then first_indent ← hang_indent
    else first_indent ← 0;
    second_width ← hsize; second_indent ← 0;
    end
  else begin first_width ← hsize; first_indent ← 0; second_width ← hsize − abs(hang_indent);
    if hang_indent ⩾ 0 then second_indent ← hang_indent
    else second_indent ← 0;
    end;
  end
```

Этот код используется в разделе 848.

850. Когда мы перейдём к следующему коду, мы как раз встретим первый активный узел r, поле *line_number* которого содержит l. Таким образом, мы хотим вычислить длину l-ой строки текущего абзаца. Более того, мы хотим установить *old_l* на последний номер в классе номеров строк эквивалентных l.

```
⟨ Рассчитай новую ширину строки 850 ⟩ ≡
  if l > easy_line then
    begin line_width ← second_width; old_l ← max_halfword − 1;
    end
  else begin old_l ← l;
    if l > last_special_line then line_width ← second_width
    else if par_shape_ptr = null then line_width ← first_width
      else line_width ← mem[par_shape_ptr + 2 ∗ l].sc;
    end
```

Этот код используется в разделе 835.

851. Оставшаяся часть процедуры *try_break* осуществляет расчёт дефектности для разрыва от r до *cur_p*.

Первое, что нужно сделать, — вычислить негодность b. Это значение всегда будет между нулём и $inf\_bad + 1$; последнее значение появляется только в строках, располагающихся от r до *cur_p*, которые не могут достаточно сжаться, чтобы вместить необходимую ширину. В таких случаях, узел r будет деактивирован. Мы также деактивируем узел r, когда разрыв на *cur_p* принудительный, т. к. будущие разрывы должны проходить через принудительный разрыв.

```
⟨ Рассмотри дефектность строки от r до cur_p; деактивируй узел r, если он не должен больше быть
      активным, затем goto continue, если строка от r до cur_p негодная, иначе запиши новый
      годный разрыв 851 ⟩ ≡
  begin artificial_demerits ← false;
  shortfall ← line_width − cur_active_width[1];   { мы считаем это слишком коротким }
  if shortfall > 0 then ⟨ Присвой b значение негодности для растяжимости строки и вычисли
      соответствующий fit_class 852 ⟩
  else ⟨ Присвой b значение негодности для сжимаемости строки и вычисли соответствующий
      fit_class 853 ⟩;
  if (b > inf_bad) ∨ (pi = eject_penalty) then ⟨ Готовься дезактивировать узел r и goto deactivate, если
      нет смысла рассматривать строки текста от r до cur_p 854 ⟩
  else begin prev_r ← r;
    if b > threshold then goto continue;
    node_r_stays_active ← true;
    end;
  ⟨ Запиши новый годный разрыв 855 ⟩;
  if node_r_stays_active then goto continue;   { prev_r устанавливается равным r }
deactivate: ⟨ Дезактивировать узел r 860 ⟩;
  end
```

Этот код используется в разделе 829.

852. Когда строка должна растягиваться, подходящая растяжимость может быть найдена в подмассиве *cur_active_width*[2 .. 5], в единицах пунктов, fil, fill и filll.

Данный раздел является частью внутреннего цикла TEX'а, и он наиболее часто выполняется, когда негодность равна бесконечности; следовательно, стоит сделать быструю проверку на наличие слишком большой ширины и маленькой растяжимости, перед вызовом подпрограммы *badness*.

```
⟨ Присвой b значение негодности для растяжимости строки и вычисли соответствующий fit_class 852
      ⟩ ≡
  if (cur_active_width[3] ≠ 0) ∨ (cur_active_width[4] ≠ 0) ∨ (cur_active_width[5] ≠ 0) then
    begin b ← 0; fit_class ← decent_fit;   { бесконечная растяжимость }
    end
  else begin if shortfall > 7230584 then
      if cur_active_width[2] < 1663497 then
        begin b ← inf_bad; fit_class ← very_loose_fit; goto done1;
        end;
    b ← badness(shortfall, cur_active_width[2]);
    if b > 12 then
      if b > 99 then fit_class ← very_loose_fit
      else fit_class ← loose_fit
    else fit_class ← decent_fit;
  done1: end
```

Этот код используется в разделе 851.

853. Сжимаемость никогда не является бесконечной в абзаце; мы можем сжать строку от r до *cur_p* самое большее на *cur_active_width*[6].

⟨ Присвой b значение негодности для сжимаемости строки и вычисли соответствующий *fit_class* 853 ⟩ ≡
```
begin if −shortfall > cur_active_width[6] then b ← inf_bad + 1
else b ← badness(−shortfall, cur_active_width[6]);
if b > 12 then fit_class ← tight_fit else fit_class ← decent_fit;
end
```
Этот код используется в разделе 851.

854. Во время последнего прохода мы осмелимся не избавляться от всех активных узлов, чтобы не потерять связь с уже найденными разрывами строк. Приведённый здесь код заботится о том, чтобы такая катастрофа не произошла, разрешая переполнение рамок, как последнее средство. Эта заслуживающая особого внимания часть TEX'а послужила источником нескольких едва уловимых ошибок, перед тем как окончательно нашли правильную логику программы; поэтому читателям, которые ищут способы «улучшить» TEX, следует трижды подумать, перед тем как осмелиться сделать какие-либо изменения здесь.

⟨ Готовься дезактивировать узел r и **goto** *deactivate*, если нет смысла рассматривать строки текста от r до *cur_p* 854 ⟩ ≡
```
begin if final_pass ∧ (minimum_demerits = awful_bad) ∧ (link(r) = last_active) ∧ (prev_r = active) then
  artificial_demerits ← true  { установить дефектность равной нулю, это вынуждает делать разрыв }
else if b > threshold then goto deactivate;
node_r_stays_active ← false;
end
```
Этот код используется в разделе 851.

855. Когда мы доходим до этой части кода, строка от r до *cur_p* является допустимой, её негодность равна b, а её класс годности — *fit_class*. Мы пока ещё не хотим создавать активный узел для этого разрыва, но мы вычислим общую дефектность и запишем их в массив *minimal_demerits*, если такой разрыв является текущим чемпионом среди всех способов дойти до *cur_p* в данном классе номеров строк и классе годности.

⟨ Запиши новый годный разрыв 855 ⟩ ≡
```
if artificial_demerits then d ← 0
else ⟨ Вычисли дефектность d от r до cur_p 859 ⟩;
stat if tracing_paragraphs > 0 then ⟨ Печатай символьное описание этого годного разрыва 856 ⟩;
tats
d ← d + total_demerits(r);  { это минимальная общая дефектность от начала до cur_p через r }
if d ⩽ minimal_demerits[fit_class] then
  begin minimal_demerits[fit_class] ← d; best_place[fit_class] ← break_node(r); best_pl_line[fit_class] ← l;
  if d < minimum_demerits then minimum_demerits ← d;
  end
```
Этот код используется в разделе 851.

856. ⟨ Печатай символьное описание этого годного разрыва 856 ⟩ ≡

```
begin if printed_node ≠ cur_p then
  ⟨ Печатай список между printed_node и cur_p, затем установи printed_node ← cur_p 857 ⟩;
print_nl("@");
if cur_p = null then print_esc("par")
else if type(cur_p) ≠ glue_node then
    begin if type(cur_p) = penalty_node then print_esc("penalty")
    else if type(cur_p) = disc_node then print_esc("discretionary")
      else if type(cur_p) = kern_node then print_esc("kern")
        else print_esc("math");
    end;
print("␣via␣@@");
if break_node(r) = null then print_char("0")
else print_int(serial(break_node(r)));
print("␣b=");
if b > inf_bad then print_char("*") else print_int(b);
print("␣p="); print_int(pi); print("␣d=");
if artificial_demerits then print_char("*") else print_int(d);
end
```

Этот код используется в разделе 855.

857. ⟨ Печатай список между *printed_node* и *cur_p*, затем установи *printed_node* ← *cur_p* 857 ⟩ ≡

```
begin print_nl("");
if cur_p = null then short_display(link(printed_node))
else begin save_link ← link(cur_p); link(cur_p) ← null; print_nl("");
  short_display(link(printed_node)); link(cur_p) ← save_link;
  end;
printed_node ← cur_p;
end
```

Этот код используется в разделе 856.

858. Когда данные для разрыва по усмотрению отображаются, мы будем должны напечатать списки *pre_break* и *post_break*; мы хотим пропустить третий список, так, чтобы данные переноса по усмотрению не появлялись дважды. Следующий код выполняется в самом конце процедуры *try_break*.

⟨ Обнови значение *printed_node* для символьного отображения 858 ⟩ ≡

```
if cur_p = printed_node then
  if cur_p ≠ null then
    if type(cur_p) = disc_node then
      begin t ← replace_count(cur_p);
      while t > 0 do
        begin decr(t); printed_node ← link(printed_node);
        end;
      end
```

Этот код используется в разделе 829.

859. ⟨ Вычисли дефектность d от r до cur_p 859 ⟩ ≡

```
begin d ← line_penalty + b;
if abs(d) ⩾ 10000 then d ← 100000000 else d ← d * d;
if pi ≠ 0 then
  if pi > 0 then d ← d + pi * pi
  else if pi > eject_penalty then d ← d − pi * pi;
if (break_type = hyphenated) ∧ (type(r) = hyphenated) then
  if cur_p ≠ null then d ← d + double_hyphen_demerits
  else d ← d + final_hyphen_demerits;
if abs(fit_class − fitness(r)) > 1 then d ← d + adj_demerits;
end
```

Этот код используется в разделе 855.

860. Когда исчезает активный узел, мы должны удалить смежный delta узел, если активный узел был вначале или в конце активного списка, или если он был окружён delta узлами. Мы также должны сохранить то свойство, что cur_active_width представляет длину материала от $link(prev\_r)$ до cur_p.

```
define combine_two_deltas(#) ≡ mem[prev_r + #].sc ← mem[prev_r + #].sc + mem[r + #].sc
define downdate_width(#) ≡ cur_active_width[#] ← cur_active_width[#] − mem[prev_r + #].sc
```

⟨ Дезактивировать узел r 860 ⟩ ≡

```
link(prev_r) ← link(r); free_node(r, active_node_size);
if prev_r = active then ⟨ Обнови активные ширины, т. к. первый активный узел удалён 861 ⟩
else if type(prev_r) = delta_node then
    begin r ← link(prev_r);
    if r = last_active then
      begin do_all_six(downdate_width); link(prev_prev_r) ← last_active;
      free_node(prev_r, delta_node_size); prev_r ← prev_prev_r;
      end
    else if type(r) = delta_node then
        begin do_all_six(update_width); do_all_six(combine_two_deltas); link(prev_r) ← link(r);
        free_node(r, delta_node_size);
        end;
    end
```

Этот код используется в разделе 851.

861. Следующий код использует то, что $type(last\_active) ≠ delta\_node$. Если активный список только что стал пустым, нам не нужно обновлять массив $active\_width$, т. к. его начальная установка выполнится, когда будет вставляться следующий активный узел.

```
define update_active(#) ≡ active_width[#] ← active_width[#] + mem[r + #].sc
```

⟨ Обнови активные ширины, т. к. первый активный узел удалён 861 ⟩ ≡

```
begin r ← link(active);
if type(r) = delta_node then
  begin do_all_six(update_active); do_all_six(copy_to_cur_active); link(active) ← link(r);
  free_node(r, delta_node_size);
  end;
end
```

Этот код используется в разделе 860.

900. После переноса слов. [Post-hyphenation] Если перенос может быть вставлен между $hc[j]$ и $hc[j+1]$, процедура переноса установит $hyf[j]$ равным некоторому небольшому нечётному числу. Но перед тем, как мы обратимся к процедуре расстановки переносов TEX'а, которая независит от остального алгоритма разрыва строк, давайте, посмотрим, что мы будем делать с точками переносов, которые она найдёт, т. к. лучше всего работать над этой частью программы, пока не забыли, чем являются ha, hb и т. д.

⟨ Общие переменные 13 ⟩ +≡
hyf: **array** [0 .. 64] **of** 0 .. 9; { нечётные значения указывают переносы по усмотрению }
init_list: *pointer*; { список знаков пунктуации, предшествующих слову }
init_lig: *boolean*; { *init_list* представляет лигатуру? }
init_lft: *boolean*; { если да, лигатура включает левую границу? }

901. ⟨ Локальные переменные для переносов 901 ⟩ ≡
i, *j*, *l*: 0 .. 65; { индексы в *hc* или *hu* }
q, *r*, *s*: *pointer*; { временные регистры для обработки списков }
bchar: *halfword*; { правый ограничительный символ переносимого слова или *non_char* }

См. также разделы 912, 922 и 929.

Этот код используется в разделе 895.

902. TEX никогда не вставит перенос, который имеет меньше, чем `\lefthyphenmin` букв, перед ним, или меньше, чем `\righthyphenmin`, букв после него. Следовательно, короткое слово имеет относительно небольшой шанс быть перенесённым. Если переносы не найдены, мы можем сберечь время, не делая ни каких изменений в абзаце.

⟨ Если не найдено переносов, **return** 902 ⟩ ≡
```
  for j ← l_hyf to hn − r_hyf do
    if odd(hyf[j]) then goto found1;
  return;
found1:
```

Этот код используется в разделе 895.

903. Если переносы на самом деле должны вставляться, TEX сперва удаляет подпоследовательность узлов между *ha* и *hb*. Делается попытка сохранить влияние, оказанное неявными ограничивающими символами и знаками пунктуации на лигатуры внутри переносимого слова, сохраняя левую границу или предшествующий символ в *hu*[0], и сохраняя возможную правую границу в *bchar*. Мы устанавливаем $j \leftarrow 0$, если *hu*[0] должна быть частью перестройки [слова]; в противном случае $j \leftarrow 1$. Переменная *s* будет указывать на остаток текущего горизонтального списка, а *q* будет указывать на узел, следующий за *hb*, чтобы можно было соединить элементы после того, как мы перестроим перенесённое по слогам слово.

```
⟨ Замени узлы ha .. hb последовательностью узлов, включающих переносы по усмотрению 903 ⟩ ≡
  q ← link(hb); link(hb) ← null; r ← link(ha); link(ha) ← null; bchar ← hyf_bchar;
  if is_char_node(ha) then
    if font(ha) ≠ hf then goto found2
    else begin init_list ← ha; init_lig ← false; hu[0] ← qo(character(ha));
      end
  else if type(ha) = ligature_node then
      if font(lig_char(ha)) ≠ hf then goto found2
      else begin init_list ← lig_ptr(ha); init_lig ← true; init_lft ← (subtype(ha) > 1);
        hu[0] ← qo(character(lig_char(ha)));
        if init_list = null then
          if init_lft then
            begin hu[0] ← 256; init_lig ← false;
            end; { в этом случае лигатура будет перестраиваться с самого начала }
        free_node(ha, small_node_size);
        end
    else begin { не найдено знаков препинания; искать левую границу }
      if ¬is_char_node(r) then
        if type(r) = ligature_node then
          if subtype(r) > 1 then goto found2;
      j ← 1; s ← ha; init_list ← null; goto common_ending;
      end;
  s ← cur_p; { мы имеем cur_p ≠ ha, потому что type(cur_p) = glue_node }
  while link(s) ≠ ha do s ← link(s);
  j ← 0; goto common_ending;
found2: s ← ha; j ← 0; hu[0] ← 256; init_lig ← false; init_list ← null;
common_ending: flush_node_list(r);
  ⟨ Восстанови узлы для переносимого слова, вставляя переносы по усмотрению 913 ⟩;
  flush_list(init_list)
```

Этот код используется в разделе 895.

904. Сейчас мы должны обратиться лицом к тому обстоятельству, что битва не закончена, даже если переносы найдены: процесс перестройки слова может быть непростым, потому что лигатуры при наличии переноса могут изменяться. В *The TEXbook* обсуждаются сложности слова "difficult", и данные переноса по усмотрению, окружающие знак переноса [дефис — прим. перев.], могут быть значительно сложнее этих. Предположим, что `abcdef` является словом, для которого в шрифте есть только лигатуры `bc`, `cd`, `de` и `ef`. Если это слово допускает перенос между `b` и `c`, есть два образца с переносом и без него `a b - cd ef` и `a bc de f`. Таким образом, вставка знака переноса может привести к «колыханию», распространяющемуся произвольно далёко в остальной части слова. Большее осложнение возникает, если вместе с таким «колыханием» появляются дополнительные переносы, например, если слово в только что приведённом примере также могло быть перенесено между `c` и `d`; TEX избегает этого, просто игнорируя дополнительные переносы в таких причудливых случаях.

Ещё бóльшие сложности возникают в присутствии лигатур, не удаляющих исходные символы. Когда знак препинания предшествует переносимому слову, способ TEX'а оказывается не таким безупречным для всех возможных сценариев, потому что знаки препинания и буквы могут распространять данные назад и вперёд. Например, предположим, исходная пара до переноса `*a` заменяется на `*y` через лигатуру `|=:`, которая заменяется на `xy` через лигатуру `=:|`; если p_{a-1} = `x` и p_a = `y`, процедура перестройки не достаточно умна, чтобы получить снова `xy`. В таких случаях разработчик шрифта должен включать лигатуру, которая переходит от `xa` к `xy`.

905. Обработке помогает функция *reconstitute*. Пусть дана строка символов $x_j \dots x_n$, и есть наименьший индекс $m \geq j$ такой, что «перевод» строки $x_j \dots x_n$ лигатурами и кернами имеет вид $y_1 \dots y_t$, за которым следует перевод строки $x_{m+1} \dots x_n$, где $y_1 \dots y_t$ — некоторая непустая последовательность узлов символов, лигатур и кернов. Мы называем $x_j \dots x_m$ «отбрасываемой приставкой» ["cut prefix"] [строки] $x_j \dots x_n$. Например, если $x_1x_2x_3$ = `fly`, и если шрифт содержит 'fl' как лигатуру и содержит керн между 'fl' и 'y', то $m = 2$, $t = 2$, а y_1 будет узлом лигатуры для 'fl', за которым следует подходящий узел керна y_2. В наиболее частом случае x_j не образует лигатуру с x_{j+1}, и мы просто имеем $m = j$, $y_1 = x_j$. Если $m < n$, мы можем повторять процедуру над $x_{m+1} \dots x_n$, пока не найдём весь перевод.

Функция *reconstitute* возвращает целое m и помещает узлы $y_1 \dots y_t$ в связанный список, начинающийся на *link*(*hold_head*), получая входные данные $x_j \dots x_n$ из массива *hu*. Если $x_j = 256$, мы считаем, что x_j должен быть неявным левым ограничительным символом; в этом случае j должен быть строго меньше, чем n. Есть параметр *bchar*, который или равен 256, или равен неявному правому ограничительному символу, который подразумевается следующим сразу за x_n. (Значение *hu*$[n + 1]$ никогда явно не проверяется, но алгоритм полагает, что *bchar* есть [в этом месте].)

Если есть индекс k в пределах $j \leq k \leq m$, такой, что *hyf* $[k]$ нечётно, и такой, что результат работы процедуры *reconstitute* был бы иным, если бы элемент x_{k+1} был *hchar*, то процедура *reconstitute* устанавливает *hyphen_passed* в наименьшее такое k. В противном случае она устанавливает *hyphen_passed* равным нулю.

Особое соглашение используется для случая $j = 0$: для него мы предполагаем, что перевод [элемента] *hu*[0] появляется в особом списке узлов символов, начинающемся на *init_list*; более того, если переменная *init_lig* установлена в *true*, то *hu*[0] будет символом лигатуры, включающим левую границу, если *init_lft* установлена в *true*. Эта возможность предназначена для случаев, когда переносимому по слогам слову предшествуют знаки пунктуации (например, одиночная или двойная кавычка), которая может влиять на перевод начала слова.

⟨ Общие переменные 13 ⟩ +≡
hyphen_passed: *small_number*; { первый знак переноса в лигатуре, если есть }

906. ⟨ Объяви функцию *reconstitute* 906 ⟩ ≡

```
function reconstitute(j,n : small_number; bchar,hchar : halfword): small_number;
  label continue,done;
  var p: pointer; { временный регистр для обработки списка }
    t: pointer; { присоединяемый узел }
    q: four_quarters; { данные символа или команда лигатуры/керна }
    cur_rh: halfword; { символ переноса [дефиса] для проверки лигатуры }
    test_char: halfword; { перенос [дефис] или другой символ для проверки лигатуры }
    w: scaled; { величина керна }
    k: font_index; { положение текущей команды лигатуры/керна }
  begin hyphen_passed ← 0; t ← hold_head; w ← 0; link(hold_head) ← null;
      { в этой точке ligature_present = lft_hit = rt_hit = false }
  ⟨ Установи структуры данных с курсором, следующим за позицией j 908 ⟩;
continue: ⟨ Если есть лигатура или керн в позиции курсора, обнови структуры данных, возможно,
      добавив j; продолжай, пока курсор не переместится 909 ⟩;
  ⟨ Добавь лигатуру и/или керн к переводу; goto continue, если стек вставленных лигатур непуст 910
      ⟩;
  reconstitute ← j;
  end;
```

Этот код используется в разделе 895.

907. Процедура перестройки использует многие глобальные структуры данных, с помощью которых TEX обработал слова, перед тем, как перенести их по слогам. Есть неявный «курсор» между символами *cur_l* и *cur_r*; эти символы будут проверяться на возможную деятельность лигатур. Если *ligature_present* [установлена, лигатура присутствует], тогда *cur_l* является лигатурой, образованной из исходных символов, следующих за *cur_q* в текущем списке перевода [translation list]. Имеется «стек лигатур» между курсором и символом $j+1$, состоящий из узлов псевдолигатур [pseudo-ligature nodes], связанных вместе их полями *link*. Этот стек обычно пуст, если команда лигатуры не создала новый символ, который нужно обработать позднее. Псевдолигатура [pseudo-ligature] — особый узел, у которого есть поле *character*, представляющее возможную лигатуру, и поле *lig_ptr*, которое или указывает на *char_node*, или содержит *null*. Мы имеем

$$cur\_r = \begin{cases} character(lig\_stack), & \text{если } lig\_stack > null; \\ qi(hu[j+1]), & \text{если } lig\_stack = null \text{ и } j < n; \\ bchar, & \text{если } lig\_stack = null \text{ и } j = n. \end{cases}$$

```
⟨ Общие переменные 13 ⟩ +≡
cur_l,cur_r: halfword; { символы перед и после курсора }
cur_q: pointer; { где лигатуру следует отделить }
lig_stack: pointer; { незаконченное дело с права от курсора }
ligature_present: boolean; { следует создавать узел лигатуры для cur_l? }
lft_hit,rt_hit: boolean; { мы попали на лигатуру с ограничительным символом? }
```

908. **define** *append_charnode_to_t*(#) ≡
begin *link*(*t*) ← *get_avail*; *t* ← *link*(*t*); *font*(*t*) ← *hf*; *character*(*t*) ← #;
end
define *set_cur_r* ≡
begin if *j* < *n* **then** *cur_r* ← *qi*(*hu*[*j* + 1]) **else** *cur_r* ← *bchar*;
if *odd*(*hyf*[*j*]) **then** *cur_rh* ← *hchar* **else** *cur_rh* ← *non_char*;
end

⟨ Установи структуры данных с курсором, следующим за позицией *j* 908 ⟩ ≡
cur_l ← *qi*(*hu*[*j*]); *cur_q* ← *t*;
if *j* = 0 **then**
begin *ligature_present* ← *init_lig*; *p* ← *init_list*;
if *ligature_present* **then** *lft_hit* ← *init_lft*;
while *p* > *null* **do**
begin *append_charnode_to_t*(*character*(*p*)); *p* ← *link*(*p*);
end;
end
else if *cur_l* < *non_char* **then** *append_charnode_to_t*(*cur_l*);
lig_stack ← *null*; *set_cur_r*

Этот код используется в разделе 906.

909. Мы можем захотеть просмотреть программу лигатур/кернов дважды, один раз для переноса, а второй для обычной буквы. (Перенос мог появиться после буквы в этой программе, поэтому лучше не пытаться искать сразу для обоих случаев.)

```
⟨ Если есть лигатура или керн в позиции курсора, обнови структуры данных, возможно, добавив j;
      продолжай, пока курсор не переместится 909 ⟩ ≡
  if cur_l = non_char then
    begin k ← bchar_label[hf];
    if k = non_address then goto done else q ← font_info[k].qqqq;
    end
  else begin q ← char_info(hf)(cur_l);
    if char_tag(q) ≠ lig_tag then goto done;
    k ← lig_kern_start(hf)(q); q ← font_info[k].qqqq;
    if skip_byte(q) > stop_flag then
      begin k ← lig_kern_restart(hf)(q); q ← font_info[k].qqqq;
      end;
    end; { теперь k — начальный адрес программы лигатур/кернов }
  if cur_rh < non_char then test_char ← cur_rh else test_char ← cur_r;
  loop begin if next_char(q) = test_char then
      if skip_byte(q) ⩽ stop_flag then
        if cur_rh < non_char then
          begin hyphen_passed ← j; hchar ← non_char; cur_rh ← non_char; goto continue;
          end
        else begin if hchar < non_char then
            if odd(hyf[j]) then
              begin hyphen_passed ← j; hchar ← non_char;
              end;
          if op_byte(q) < kern_flag then ⟨ Выполни замену лигатур, обновляя структуру курсора и,
                возможно, увеличивая j; goto continue если курсор не перемещается, иначе goto
                done 911 ⟩;
          w ← char_kern(hf)(q); goto done; { этот керн вставим ниже }
          end;
    if skip_byte(q) ⩾ stop_flag then
      if cur_rh = non_char then goto done
      else begin cur_rh ← non_char; goto continue;
        end;
    k ← k + qo(skip_byte(q)) + 1; q ← font_info[k].qqqq;
    end;
done:
```

Этот код используется в разделе 906.

910. **define** *wrap_lig*(#) ≡
  if *ligature_present* **then**
    begin $p \leftarrow$ *new_ligature*(*hf*, *cur_l*, *link*(*cur_q*));
    if *lft_hit* **then**
      begin *subtype*(*p*) ← 2; *lft_hit* ← *false*;
      end;
    if # **then**
      if *lig_stack* = *null* **then**
        begin *incr*(*subtype*(*p*)); *rt_hit* ← *false*;
        end;
    link(*cur_q*) ← *p*; *t* ← *p*; *ligature_present* ← *false*;
    end
define *pop_lig_stack* ≡
  begin if *lig_ptr*(*lig_stack*) > *null* **then**
    begin *link*(*t*) ← *lig_ptr*(*lig_stack*); { это — узел символа для $hu[j+1]$ }
    t ← *link*(*t*); *incr*(*j*);
    end;
  p ← *lig_stack*; *lig_stack* ← *link*(*p*); *free_node*(*p*, *small_node_size*);
  if *lig_stack* = *null* **then** *set_cur_r* **else** *cur_r* ← *character*(*lig_stack*);
  end { если *lig_stack* не равен *null*, мы имеем *cur_rh* = *non_char* }

⟨ Добавь лигатуру и/или керн к переводу; **goto** *continue*, если стек вставленных лигатур непуст 910 ⟩ ≡
  wrap_lig(*rt_hit*);
  if $w \neq 0$ **then**
    begin *link*(*t*) ← *new_kern*(*w*); *t* ← *link*(*t*); *w* ← 0;
    end;
  if *lig_stack* > *null* **then**
    begin *cur_q* ← *t*; *cur_l* ← *character*(*lig_stack*); *ligature_present* ← *true*; *pop_lig_stack*; **goto** *continue*;
    end

Этот код используется в разделе 906.

911. ⟨ Выполни замену лигатур, обновляя структуру курсора и, возможно, увеличивая j; **goto** *continue* если курсор не перемещается, иначе **goto** *done* 911 ⟩ ≡

```
begin if cur_l = non_char then lft_hit ← true;
if j = n then
  if lig_stack = null then rt_hit ← true;
check_interrupt; { позволим выйти, если есть бесконечная петля лигатур }
case op_byte(q) of
qi(1), qi(5): begin cur_l ← rem_byte(q); { =:|, =:|> }
  ligature_present ← true;
  end;
qi(2), qi(6): begin cur_r ← rem_byte(q); { |=:, |=:> }
  if lig_stack > null then character(lig_stack) ← cur_r
  else begin lig_stack ← new_lig_item(cur_r);
    if j = n then bchar ← non_char
    else begin p ← get_avail; lig_ptr(lig_stack) ← p; character(p) ← qi(hu[j + 1]); font(p) ← hf;
      end;
    end;
  end;
qi(3): begin cur_r ← rem_byte(q); { |=:| }
  p ← lig_stack; lig_stack ← new_lig_item(cur_r); link(lig_stack) ← p;
  end;
qi(7), qi(11): begin wrap_lig(false); { |=:|>, |=:|>> }
  cur_q ← t; cur_l ← rem_byte(q); ligature_present ← true;
  end;
othercases begin cur_l ← rem_byte(q); ligature_present ← true; { =: }
  if lig_stack > null then pop_lig_stack
  else if j = n then goto done
    else begin append_charnode_to_t(cur_r); incr(j); set_cur_r;
      end;
  end
endcases;
if op_byte(q) > qi(4) then
  if op_byte(q) ≠ qi(7) then goto done;
goto continue;
end
```

Этот код используется в разделе 909.

912. Хорошо, мы готовы вставить возможные переносы, которые нашли. Когда выполняется следующая программа, мы хотим добавить слово $hu[1 \,..\, hn]$ после узла ha, и узел q должен присоединяться к результату. Во время этого процесса, переменная i будет временным индексом в hu; переменная j будет индексом нашего текущего положения в hu; переменная l будет образом [counterpart] переменной j в ветви переносов [discretionary branch]; переменная r будет указывать на новые создаваемые узлы; и нам нужно несколько локальных переменных:

⟨ Локальные переменные для переносов 901 ⟩ +≡

```
major_tail, minor_tail: pointer;
    { концы списков в главной и разбиваемой переносами перестраиваемых ветвях }
c: ASCII_code; { символ, временно замещённый переносом }
c_loc: 0 .. 63; { откуда этот символ поступил }
r_count: integer; { счётчик замен для переносов по слогам }
hyf_node: pointer; { перенос [hyphen], если он существует }
```

913. Когда выполняется следующий код, $hyf[0]$ и $hyf[hn]$ будут равны нулю.

```
⟨ Восстанови узлы для переносимого слова, вставляя переносы по усмотрению 913 ⟩ ≡
  repeat l ← j; j ← reconstitute(j, hn, bchar, qi(hyf_char)) + 1;
    if hyphen_passed = 0 then
      begin link(s) ← link(hold_head);
      while link(s) > null do s ← link(s);
      if odd(hyf[j − 1]) then
        begin l ← j; hyphen_passed ← j − 1; link(hold_head) ← null;
        end;
      end;
    if hyphen_passed > 0 then
      ⟨ Создай и добавь узел переноса по усмотрению, как альтернативу неперенесённому слову и
          продолжи развивать обе ветви, пока они не станут равноценны 914 ⟩;
  until j > hn;
  link(s) ← q
```

Этот код используется в разделе 903.

914. В этом повторяющемся цикле мы вставим другой знак переноса, если $hyf[j-1]$ — нечётное, когда обе ветви предыдущего переноса по усмотрению заканчиваются в положении $j-1$. Строго говоря, мы не имеем всех оснований делать это, потому что мы не знаем, действительно ли перенос после $j-1$ не зависит от этих ветвей. Но почти во всех приложениях мы не потеряли бы возможно значимую точку переноса. (Рассмотрите слово 'difficult', где буква 'c' находится в положении j.)

```
  define advance_major_tail ≡
           begin major_tail ← link(major_tail); incr(r_count);
           end
```

```
⟨ Создай и добавь узел переноса по усмотрению, как альтернативу неперенесённому слову и продолжи
    развивать обе ветви, пока они не станут равноценны 914 ⟩ ≡
  repeat r ← get_node(small_node_size); link(r) ← link(hold_head); type(r) ← disc_node; major_tail ← r;
    r_count ← 0;
    while link(major_tail) > null do advance_major_tail;
    i ← hyphen_passed; hyf[i] ← 0; ⟨ Положи символы hu[l .. i] и знак переноса в pre_break(r) 915 ⟩;
    ⟨ Помести символы hu[i + 1 ..] в post_break(r), добавляя к этому списку и к major_tail, пока не
        будет достигнута синхронизация 916 ⟩;
    ⟨ Перемести указатель s на конец текущего списка и установи replace_count(r) надлежащим
        образом 918 ⟩;
    hyphen_passed ← j − 1; link(hold_head) ← null;
  until ¬odd(hyf[j − 1])
```

Этот код используется в разделе 913.

915. Новый перенос [дефис, hyphen] мог объединиться с предыдущим символом через лигатуру или керн. В этом месте мы имеем $l-1 \leqslant i < j$ и $i < hn$.

```
⟨ Положи символы hu[l .. i] и знак переноса в pre_break(r) 915 ⟩ ≡
  minor_tail ← null; pre_break(r) ← null; hyf_node ← new_character(hf, hyf_char);
  if hyf_node ≠ null then
    begin incr(i); c ← hu[i]; hu[i] ← hyf_char; free_avail(hyf_node);
    end;
  while l ⩽ i do
    begin l ← reconstitute(l, i, font_bchar[hf], non_char) + 1;
    if link(hold_head) > null then
      begin if minor_tail = null then pre_break(r) ← link(hold_head)
      else link(minor_tail) ← link(hold_head);
      minor_tail ← link(hold_head);
      while link(minor_tail) > null do minor_tail ← link(minor_tail);
      end;
    end;
  if hyf_node ≠ null then
    begin hu[i] ← c; { восстановить символ в месте переноса }
    l ← i; decr(i);
    end
```

Этот код используется в разделе 914.

916. Алгоритм синхронизации начинает с $l = i + 1 \leqslant j$.

```
⟨ Помести символы hu[i + 1 .. ] в post_break(r), добавляя к этому списку и к major_tail, пока не будет
      достигнута синхронизация 916 ⟩ ≡
  minor_tail ← null; post_break(r) ← null; c_loc ← 0;
  if bchar_label[hf] ≠ non_address then { поместить левую границу вначале новой строки }
    begin decr(l); c ← hu[l]; c_loc ← l; hu[l] ← 256;
    end;
  while l < j do
    begin repeat l ← reconstitute(l, hn, bchar, non_char) + 1;
      if c_loc > 0 then
        begin hu[c_loc] ← c; c_loc ← 0;
        end;
      if link(hold_head) > null then
        begin if minor_tail = null then post_break(r) ← link(hold_head)
        else link(minor_tail) ← link(hold_head);
        minor_tail ← link(hold_head);
        while link(minor_tail) > null do minor_tail ← link(minor_tail);
        end;
    until l ⩾ j;
    while l > j do ⟨ Добавь символы от hu[j .. ] до major_tail и переместись вперёд на j 917 ⟩;
    end
```

Этот код используется в разделе 914.

917.

```
⟨ Добавь символы от hu[j .. ] до major_tail и переместись вперёд на j 917 ⟩ ≡
  begin j ← reconstitute(j, hn, bchar, non_char) + 1; link(major_tail) ← link(hold_head);
  while link(major_tail) > null do advance_major_tail;
  end
```

Этот код используется в разделе 916.

918. Вставка лигатур может стать причиной того, что размер слова начнёт экспоненциально расти. Поэтому мы должны проверить здесь размер *r_count*, даже если переносимый текст был не более 63 символов длиной.

```
⟨ Перемести указатель s на конец текущего списка и установи replace_count(r) надлежащим образом 918
      ⟩ ≡
  if r_count > 127 then { мы должны забыть перенос по усмотрению }
    begin link(s) ← link(r); link(r) ← null; flush_node_list(r);
    end
  else begin link(s) ← r; replace_count(r) ← r_count;
    end;
  s ← major_tail
```

Этот код используется в разделе 914.

1380. Указатель. [Index] Здесь вы можете для каждого идентификатора найти все те места в программе, где он использован. Подчёркнутые номера указывают, где он определён. Для идентификаторов длиной всего в одну букву Вы увидите только подчёркнутые номера. *Все ссылки являются номерами разделов вместо номеров страниц.*

В этом указателе также перечислены сообщения об ошибках и другие стороны программы, которые Вы можете захотеть когда-нибудь изучить. Например, в записи «системные зависимости» [“system dependencies”] перечислены все разделы, требующие особого внимания людей, устанавливающих TEX в новом операционном окружении. Перечень различных вещей, которые не могут произойти, приведён в разделе «это не может происходить» [“this can’t happen”]. Примерно 40 разделов перечисленно в записи «внутренний цикл» [“inner loop”]; они занимают около 60% времени выполнения TEX’a, не считая ввода и вывода.

⟨ em-ширина для *cur_font* 558 ⟩ Используется в разделе 455.
⟨ x-высота для *cur_font* 559 ⟩ Используется в разделе 455.
⟨ Введи *skip_blanks* состояние, выдай пробел 349 ⟩ Используется в разделе 347.
⟨ Введи все перечисленные исключения из правил переноса, пока не встретишь правую скобку; затем **return** 935 ⟩ Используется в разделе 934.
⟨ Введи для `\read` с терминала 484 ⟩ Используется в разделе 483.
⟨ Введи и сохрани лексемы из следующей строки файла 483 ⟩ Используется в разделе 482.
⟨ Введи из внешнего файла, **goto** *restart* если вводить нечего 343 ⟩ Используется в разделе 341.
⟨ Введи из списка лексем, **goto** *restart*, если конец списка или если параметра нужно раскрыть 357 ⟩ Используется в разделе 341.
⟨ Введи исключение переноса 939 ⟩ Используется в разделе 935.
⟨ Введи первую строку из *read_file*[*m*] 485 ⟩ Используется в разделе 483.
⟨ Введи следующую строку из *read_file*[*m*] 486 ⟩ Используется в разделе 483.
⟨ Вводи все шаблоны в связное дерево trie, пока не дойдёшь до правой фигурной скобки 961 ⟩ Используется в разделе 960.
⟨ Включи размеры рамки в размеры hbox, которая будет её содержать 653 ⟩ Используется в разделе 651.
⟨ Включи размеры рамки в размеры vbox, которая будет её содержать 670 ⟩ Используется в разделе 669.
⟨ Включи размеры символа в размеры hbox, которая будет его содержать, затем перейди к следующему узлу 654 ⟩ Используется в разделе 651.
⟨ Включи узел «нечто» в hbox 1360 ⟩ Используется в разделе 651.
⟨ Включи узел «нечто» в vbox 1359 ⟩ Используется в разделе 669.
⟨ Внеси всю группу в текущий параметр 399 ⟩ Используется в разделе 392.
⟨ Внеси недавно подобранные лексемы в текущий параметр, и **goto** *continue*, если частичное соответствие ещё действует; но прервись, если $s = null$ 397 ⟩ Используется в разделе 392.
⟨ Возобнови построение страниц после того, как программа вывода завершится 1026 ⟩ Используется в разделе 1100.
⟨ Возьми из стека условий 496 ⟩ Используется в разделах 498, 500, 509 и 510.
⟨ Возьми первую строку ввода и готовься начать 1337 ⟩ Используется в разделе 1332.
⟨ Возьми следующую лексему, которая не пустая, не relax и не вызов 404 ⟩ Используется в разделах 403, 1078, 1084, 1151, 1160, 1211, 1226 и 1270.
⟨ Возьми следующую лексему, подавив раскрытие 358 ⟩ Используется в разделе 357.
⟨ Войди в режим выделенной формулы 1145 ⟩ Используется в разделе 1138.
⟨ Войди в режим обычной математики 1139 ⟩ Используется в разделах 1138 и 1142.
⟨ Восстанови внешнюю управляющую последовательность, чтобы перечитать её 337 ⟩ Используется в разделе 336.
⟨ Восстанови узлы для переносимого слова, вставляя переносы по усмотрению 913 ⟩ Используется в разделе 903.
⟨ Вставь текущий список в его окружение 812 ⟩ Используется в разделе 800.
⟨ Вставь пару (s, p) в таблицу исключений 940 ⟩ Используется в разделе 939.
⟨ Вставь $\langle v_j \rangle$ шаблон и **goto** *restart* 789 ⟩ Используется в разделе 342.
⟨ Вставь клей в вертикальные итоги 671 ⟩ Используется в разделе 669.
⟨ Вставь клей в горизонтальные итоги 656 ⟩ Используется в разделе 651.
⟨ Вставь клей для *split_top_skip* и установи $p \leftarrow null$ 969 ⟩ Используется в разделе 968.
⟨ Вставь лексему, содержащую *frozen_endv* 375 ⟩ Используется в разделе 366.
⟨ Вставь лексему, сохранённую `\afterassignment`, если есть 1269 ⟩ Используется в разделе 1211.
⟨ Вставь новую управляющую последовательность после p, затем установи p на неё 260 ⟩ Используется в разделе 259.
⟨ Вставь новые данные с терминала и **return** 87 ⟩ Используется в разделе 84.
⟨ Вставь новый активный узел от *best_place*[*fit_class*] до *cur_p* 845 ⟩ Используется в разделе 836.
⟨ Вставь новый образец в связанное дерево (trie) 963 ⟩ Используется в разделе 961.
⟨ Вставь новый узел trie, между q и p, и установи p на него 964 ⟩ Используется в разделе 963.
⟨ Вставь параметр макроса и **goto** *restart* 359 ⟩ Используется в разделе 357.

⟨ Вставь переносы, как указано в *hyph_list*[*h*] 932 ⟩ Используется в разделе 931.
⟨ Вставь узел delta, чтобы подготовить следующий активный узел 844 ⟩ Используется в разделе 836.
⟨ Вставь узел delta, чтобы подготовиться к разрыву на *cur_p* 843 ⟩ Используется в разделе 836.
⟨ Вставь узел *p* в список текущей страницы и **goto** *done* 998 ⟩ Используется в разделе 997.
⟨ Вставь фиктивный мат. узел для верхнего или нижнего индекса 1177 ⟩ Используется в разделе 1176.
⟨ Выбери подходящий случай и **return** или **goto** *common_ending* 509 ⟩ Используется в разделе 501.
⟨ Выведи заполнители в вертикальный список, **goto** *fin_rule*, если линейка, или **goto** *next_p*, если выполнено 635 ⟩ Используется в разделе 634.
⟨ Выведи заполнитель в горизонтальный список hlist, **goto** *fin_rule*, если линейка или к *next_p*, если готово 626 ⟩ Используется в разделе 625.
⟨ Выведи имя шрифта с внутренним номером *f* 603 ⟩ Используется в разделе 602.
⟨ Выведи линейку в hlist 624 ⟩ Используется в разделе 622.
⟨ Выведи линейку в vlist, **goto** *next_p* 633 ⟩ Используется в разделе 631.
⟨ Выведи не-*char_node* *p* для *hlist_out* и перейди к следующему узлу 622 ⟩ Используется в разделе 620.
⟨ Выведи не-*char_node* *p* для *vlist_out* 631 ⟩ Используется в разделе 630.
⟨ Выведи определения шрифтов для всех использованных шрифтов 643 ⟩ Используется в разделе 642.
⟨ Выведи последние байты *dvi_buf* 599 ⟩ Используется в разделе 642.
⟨ Выведи рамку в hlist 623 ⟩ Используется в разделе 622.
⟨ Выведи рамку в vlist 632 ⟩ Используется в разделе 631.
⟨ Выведи рамку заполнителя в *cur_h*, и передвинь *cur_h* на *leader_wd* + *lx* 628 ⟩ Используется в разделе 626.
⟨ Выведи рамку заполнителя в *cur_v*, и передвинь *cur_v* на *leader_ht* + *lx* 637 ⟩ Используется в разделе 635.
⟨ Выведи статистику об этой работе 1334 ⟩ Используется в разделе 1333.
⟨ Выведи узел «нечто» *p* в hlist 1367 ⟩ Используется в разделе 622.
⟨ Выведи узел «нечто» *p* в vlist 1366 ⟩ Используется в разделе 631.
⟨ Выведи узел *p* для *hlist_out* и перейди к следующему узлу, поддерживающему условие *cur_v* = *base_line* 620 ⟩ Используется в разделе 619.
⟨ Выведи узел *p* для *vlist_out* и перейди к следующему узлу, поддерживающему условие *cur_h* = *left_edge* 630 ⟩ Используется в разделе 629.
⟨ Выгрузи динамическую память 1311 ⟩ Используется в разделе 1302.
⟨ Выгрузи ещё пару вещей и завершающее контрольное слово 1326 ⟩ Используется в разделе 1302.
⟨ Выгрузи константы для проверки целостности 1307 ⟩ Используется в разделе 1302.
⟨ Выгрузи массив сведений о внутреннем шрифте номер *k* 1322 ⟩ Используется в разделе 1320.
⟨ Выгрузи области 5 и 6 массива *eqtb* 1316 ⟩ Используется в разделе 1313.
⟨ Выгрузи области с 1 по 4 массива *eqtb* 1315 ⟩ Используется в разделе 1313.
⟨ Выгрузи пул строк 1309 ⟩ Используется в разделе 1302.
⟨ Выгрузи сведения о шрифтах 1320 ⟩ Используется в разделе 1302.
⟨ Выгрузи таблицу эквивалентов 1313 ⟩ Используется в разделе 1302.
⟨ Выгрузи таблицы переносов 1324 ⟩ Используется в разделе 1302.
⟨ Выгрузи хеш-таблицу 1318 ⟩ Используется в разделе 1313.
⟨ Выдели весь узел *p* и **goto** *found* 129 ⟩ Используется в разделе 127.
⟨ Выдели из вершины узла *p* и **goto** *found* 128 ⟩ Используется в разделе 127.
⟨ Вызови *try_break*, если *cur_p* — допустимая точка разрыва; при втором проходе, также попробуй перенести следующее слово, если *cur_p* — узел клея; затем передвинь *cur_p* на следующий узел абзаца, который возможно мог бы быть допустимой точкой разрыва 866 ⟩ Используется в разделе 863.
⟨ Вызови подпрограмму упаковкие, установив *just_box* на выровненную рамку 889 ⟩ Используется в разделе 880.
⟨ Выкинь верхний уровень из *save_stack* 282 ⟩ Используется в разделе 281.
⟨ Выкинь рамку из памяти, покажи статистику если требуется 639 ⟩ Используется в разделе 638.
⟨ Выполни `\closeout` 1353 ⟩ Используется в разделе 1348.

⟨ Выполни `\immediate` 1375 ⟩ Используется в разделе 1348.
⟨ Выполни `\openout` 1351 ⟩ Используется в разделе 1348.
⟨ Выполни `\setlanguage` 1377 ⟩ Используется в разделе 1348.
⟨ Выполни `\special` 1354 ⟩ Используется в разделе 1348.
⟨ Выполни `\write` 1352 ⟩ Используется в разделе 1348.
⟨ Выполни замену лигатур, обновляя структуру курсора и, возможно, увеличивая j; **goto** *continue* если курсор не перемещается, иначе **goto** *done* 911 ⟩ Используется в разделе 909.
⟨ Выполни код c и **return**, если сделано 84 ⟩ Используется в разделе 83.
⟨ Выполни программу вывода по умолчанию 1023 ⟩ Используется в разделе 1012.
⟨ Выполни работу, стоящую в очереди для `\write` 1374 ⟩ Используется в разделе 1373.
⟨ Выровняй конец строки в точке разрыва *cur_p* и добавь её в текущий вертикальный список, вместе со связанными штрафами и другими вставками 880 ⟩ Используется в разделе 877.
⟨ Вычисли величину скоса 741 ⟩ Используется в разделе 738.
⟨ Вычисли волшебное смещение 765 ⟩ Используется в разделе 1337.
⟨ Вычисли дефектность d от r до *cur_p* 859 ⟩ Используется в разделе 855.
⟨ Вычисли длину l и величину сдвига s выделенных строк 1149 ⟩ Используется в разделе 1145.
⟨ Вычисли естественную ширину w, на которую символы последней строки простираются вправо до заданной точки (reference point) плюс два ems; или установи $w \leftarrow max\_dimen$, если на непустые данные на этой строке влияют растяжение или сжатие 1146 ⟩ Используется в разделе 1145.
⟨ Вычисли значения *break_width* 837 ⟩ Используется в разделе 836.
⟨ Вычисли значения переноса по усмотрению *break_width* 840 ⟩ Используется в разделе 837.
⟨ Вычисли итог *multiply* или *divide*, помести его в *cur_val* 1240 ⟩ Используется в разделе 1236.
⟨ Вычисли итог *register* или *advance*, помести его в *cur_val* 1238 ⟩ Используется в разделе 1236.
⟨ Вычисли код операции дерева (trie) v и установи $l \leftarrow 0$ 965 ⟩ Используется в разделе 963.
⟨ Вычисли наименьшую подходящую высоту w и соответствующее число шагов расширения n; также установи $width(b)$ 714 ⟩ Используется в разделе 713.
⟨ Вычисли негодность b текущей страницы, используя *awful_bad*, если рамка слишком полная 1007 ⟩ Используется в разделе 1005.
⟨ Вычисли негодность b, используя *awful_bad*, если рамка слишком полная 975 ⟩ Используется в разделе 974.
⟨ Вычисли положение регистра l и его тип p; но **return**, если неверно 1237 ⟩ Используется в разделе 1236.
⟨ Вычисли сумму двух спецификаций клея 1239 ⟩ Используется в разделе 1238.
⟨ Вычисли хеш-код h 261 ⟩ Используется в разделе 259.
⟨ Вычти клей из *break_width* 838 ⟩ Используется в разделе 837.
⟨ Вычти ширину узла v из *break_width* 841 ⟩ Используется в разделе 840.
⟨ Готовься вставить лексему, соответствующую *cur_group*, и напечатая, что она такое есть 1065 ⟩ Используется в разделе 1064.
⟨ Готовься дезактивировать узел r и **goto** *deactivate*, если нет смысла рассматривать строки текста от r до *cur_p* 854 ⟩ Используется в разделе 851.
⟨ Готовься к другому символу или оставь *lig_stack* пустым, если его нет 1038 ⟩ Используется в разделе 1034.
⟨ Готовься переместить «нечто» p на текущую страницу, затем **goto** *contribute* 1364 ⟩ Используется в разделе 1000.
⟨ Готовься переместить узел рамки или линейки на текущую страницу, затем **goto** *contribute* 1002 ⟩ Используется в разделе 1000.
⟨ Готовься разбить строку 816, 827, 834, 848 ⟩ Используется в разделе 815.
⟨ Готовься сжать trie 952 ⟩ Используется в разделе 966.
⟨ Дай диагностические сведения, если требуется 1031 ⟩ Используется в разделе 1030.
⟨ Дай ошибку о негодном `\hyphenation` 936 ⟩ Используется в разделе 935.
⟨ Дай совет пользователю и **return** 83 ⟩ Используется в разделе 82.
⟨ Двигайся вниз или выведи заполнитель 634 ⟩ Используется в разделе 631.
⟨ Дезактивировать узел r 860 ⟩ Используется в разделе 851.

⟨ Делай команду лигатуры или керна, возвращаясь в *main_lig_loop*, или *main_loop_wrapup*, или *main_loop_move* 1040 ⟩ Используется в разделе 1039.
⟨ Делай первый проход обработки на основе *type*(*q*); **goto** *done_with_noad* если математический узел noad полностью обработан, **goto** *check_dimensions*, если он переведён в *new_hlist*(*q*), или **goto** *done_with_node*, если узел полностью обработан 728 ⟩ Используется в разделе 727.
⟨ Директивы компилятора 9 ⟩ Используется в разделе 4.
⟨ Добавь акцент с подходящими кернами, затем установи $p \leftarrow q$ 1125 ⟩ Используется в разделе 1123.
⟨ Добавь вставку на текущую страницу и **goto** *contribute* 1008 ⟩ Используется в разделе 1000.
⟨ Добавь выделенную формулу и, возможно, номер выражения 1204 ⟩ Используется в разделе 1199.
⟨ Добавь значение *n* в список *p* 938 ⟩ Используется в разделе 937.
⟨ Добавь клей tabskip и пустую рамку к списку *u* и обнови *s* и *t*, а также принятые узлы прообразов 809 ⟩ Используется в разделе 808.
⟨ Добавь клей или номер выражения перед выделенной формулой 1203 ⟩ Используется в разделе 1199.
⟨ Добавь клей или номер выражения после выделенной формулы 1205 ⟩ Используется в разделе 1199.
⟨ Добавь лигатуру и/или керн к переводу; **goto** *continue*, если стек вставленных лигатур непуст 910 ⟩ Используется в разделе 906.
⟨ Добавь любые элементы *new_hlist* элементы для *q*, и любые подходящие штрафы 767 ⟩ Используется в разделе 760.
⟨ Добавь межэлементный промежуток на основе *r_type* и *t* 766 ⟩ Используется в разделе 760.
⟨ Добавь новую букву или перенос 937 ⟩ Используется в разделе 935.
⟨ Добавь новую букву или уровень переноса 962 ⟩ Используется в разделе 961.
⟨ Добавь новую рамку к текущему вертикальному списку, за которым следует список специальных узлов полученных из рамок упаковщиком 888 ⟩ Используется в разделе 880.
⟨ Добавь новый узел заполнителя, который использует *cur_box* 1078 ⟩ Используется в разделе 1075.
⟨ Добавь обычный межсловный промежуток в текущий список, затем **goto** *big_switch* 1041 ⟩ Используется в разделе 1030.
⟨ Добавь рамку *cur_box* в текущий список, сдвинутый на *box_context* 1076 ⟩ Используется в разделе 1075.
⟨ Добавь символ *cur_chr* и следующие символы (если есть) к текущему hlist в текущий шрифт; **goto** *reswitch*, когда встретится не-символ 1034 ⟩ Используется в разделе 1030.
⟨ Добавь символы от *hu*[*j* . .] до *major_tail* и переместись вперёд на *j* 917 ⟩ Используется в разделе 916.
⟨ Добавь текущий клей tabskip к списку преамбулы 778 ⟩ Используется в разделе 777.
⟨ Добавь узел штрафа, если ненулевой штраф присваивается 890 ⟩ Используется в разделе 880.
⟨ Добавь ширину узла *s* к *act_width* 871 ⟩ Используется в разделе 869.
⟨ Добавь ширину узла *s* к *break_width* 842 ⟩ Используется в разделе 840.
⟨ Добавь ширину узла *s* к *disc_width* 870 ⟩ Используется в разделе 869.
⟨ Добудь *dead_cycles* или *insert_penalties* 419 ⟩ Используется в разделе 413.
⟨ Добудь *prev_graf* 422 ⟩ Используется в разделе 413.
⟨ Добудь *space_factor* или *prev_depth* 418 ⟩ Используется в разделе 413.
⟨ Добудь код символа из некоторой таблицы 414 ⟩ Используется в разделе 413.
⟨ Добудь размер *par_shape* 423 ⟩ Используется в разделе 413.
⟨ Добудь размер рамки 420 ⟩ Используется в разделе 413.
⟨ Добудь размер шрифта 425 ⟩ Используется в разделе 413.
⟨ Добудь регистр 427 ⟩ Используется в разделе 413.
⟨ Добудь список лексем или идентификатор шрифта, если только *level* = *tok_val* 415 ⟩ Используется в разделе 413.
⟨ Добудь целое шрифта 426 ⟩ Используется в разделе 413.
⟨ Добудь что-нибудь на *page_so_far* 421 ⟩ Используется в разделе 413.
⟨ Добудь элемент в текущем узле, если подходит 424 ⟩ Используется в разделе 413.
⟨ Другие локальные переменные для *try_break* 830 ⟩ Используется в разделе 829.
⟨ Если все символы семейства подходят для *h*, то **goto** *found*, иначе **goto** *not_found* 955 ⟩ Используется в разделе 953.
⟨ Если выгрузка запрещена, прервись 1304 ⟩ Используется в разделе 1302.

⟨ Если есть команда лигатуры/керна, относящаяся к *cur_l* и *cur_r*, выровняй соответствующим образом текст; выйди в *main_loop_wrapup* 1039 ⟩ Используется в разделе 1034.

⟨ Если есть лигатура или керн в позиции курсора, обнови структуры данных, возможно, добавив *j*; продолжай, пока курсор не переместится 909 ⟩ Используется в разделе 906.

⟨ Если за курсором непосредственно следует правая граница, **goto** *reswitch*; если за ним следует недопустимый символ, **goto** *big_switch*; иначе передвинь курсор на один шаг вправо и **goto** *main_lig_loop* 1036 ⟩ Используется в разделе 1034.

⟨ Если инструкция *cur_i* — керн с *cur_c*, присоедини керн после *q*; или, если она — лигатура с *cur_c*, объедини узлы noads *q* и *p* соответствующим образом; затем **return**, если курсор не указывает на узел noad, или **goto** *restart* 753 ⟩ Используется в разделе 752.

⟨ Если класс номеров строк завершился, создай новые активные узлы для наилучших годных разрывов в этом классе; затем **return** если $r = last\_active$, иначе вычисли новую *line_width* 835 ⟩ Используется в разделе 829.

⟨ Если не найдено переносов, **return** 902 ⟩ Используется в разделе 895.

⟨ Если присутствует раскрываемый код, сократи его и **goto** *start_cs* 355 ⟩ Используется в разделах 354 и 356.

⟨ Если с правой стороны параметр лексемы или регистр лексемы, заверши присвоение и **goto** *done* 1227 ⟩ Используется в разделе 1226.

⟨ Если следующий символ — номер параметра, сделай *cur_tok* лексемой *match*; но если он — левая фигурная скобка, сохрани '*left_brace*, *end_match*', установи *hash_brace* и **goto** *done* 476 ⟩ Используется в разделе 474.

⟨ Если список преамбулы пройден, проверь, что строка таблицы закончилась 792 ⟩ Используется в разделе 791.

⟨ Если строка $hyph\_word[h]$ меньше или равна *s*, обменяй $(hyph\_word[h], hyph\_list[h])$ и (s, p) 941 ⟩ Используется в разделе 940.

⟨ Если строка $hyph\_word[h]$ меньше, чем $hc[1 \mathinner{..} hn]$, **goto** *not_found*; но если две строки равны, установи *hyf* равным месту переноса (hyphen) и **goto** *found* 931 ⟩ Используется в разделе 930.

⟨ Если текущая страница пуста, и узел *p* должен быть удалён, **goto** *done1*; иначе используй узел *p*, чтобы обновить состояние текущей страницы; если этот узел — вставка, **goto** *contribute*; иначе, если этот узел — недопустимая точка разрыва, **goto** *contribute* или *update_heights*; иначе установи *pi* равным штрафу, соответствующему данной точке разрыва 1000 ⟩ Используется в разделе 997.

⟨ Если текущий список закончился узлом рамки, удали его из списка и установи *cur_box* на него; иначе установи $cur\_box \leftarrow null$ 1080 ⟩ Используется в разделе 1079.

⟨ Если узел *cur_p* — допустимая точка разрыва, вызови *try_break*; затем обнови активные ширины, включив клей в $glue\_ptr(cur\_p)$ 868 ⟩ Используется в разделе 866.

⟨ Если узел *p* — допустимая точка разрыва, проверь, является ли этот разрыва лучшим известным, и **goto** *done*, если *p* пусто или если страница уже слишком полна, чтобы добавлять ещё что-нибудь 972 ⟩ Используется в разделе 970.

⟨ Если узел *q* — узел стиля, измени стиль и **goto** *delete_q*; иначе, если он не узел noad, помести его в hlist, передвинь *q* вперёд и **goto** *done*, иначе установи *s* на размер узла noad *q*, установи в *t* соответствующий тип $(ord\_noad \mathinner{..} inner\_noad)$ и установи в *pen* соответствующий штраф 761 ⟩ Используется в разделе 760.

⟨ Если узел *r* типа *delta_node*, обнови *cur_active_width*, установи *prev_r* и *prev_prev_r*, затем **goto** *continue* 832 ⟩ Используется в разделе 829.

⟨ Если элемента выравнивания только что закончился, действуй должным образом 342 ⟩ Используется в разделе 341.

⟨ Если этот *sup_mark* начал раскрывать символ вроде `^^A` или `^^df`, то **goto** *reswitch*, иначе установи $state \leftarrow mid\_line$ 352 ⟩ Используется в разделе 344.

⟨ Если этот шрифт уже загружен, установи *f* на внутренний номер шрифта и **goto** *common_ending* 1260 ⟩ Используется в разделе 1257.

⟨ Заверши `DVI` файл 642 ⟩ Используется в разделе 1333.

⟨ Заверши возможно длинную команду `\show` 1298 ⟩ Используется в разделе 1293.
⟨ Заверши выделенную формулу 1199 ⟩ Используется в разделе 1194.
⟨ Заверши выравнивание в выделенной формуле [display?] 1206 ⟩ Используется в разделе 812.
⟨ Заверши незавершённый математический узел 1185 ⟩ Используется в разделе 1184.
⟨ Заверши рамку, указанную узлом r, разбив узел p, если требуется; установи $wait \leftarrow true$, если узел p хранит оставшуюся после разделения часть 1021 ⟩ Используется в разделе 1020.
⟨ Заверши строку, **goto** *switch* 350 ⟩ Используется в разделе 347.
⟨ Заверши строку, выдай `\par` 351 ⟩ Используется в разделе 347.
⟨ Заверши строку, выдай пробел 348 ⟩ Используется в разделе 347.
⟨ Заверши текущее условие и пропусти до `\fi` 510 ⟩ Используется в разделе 367.
⟨ Заверши формулу в тексте 1196 ⟩ Используется в разделе 1194.
⟨ Завершить расширения 1378 ⟩ Используется в разделе 1333.
⟨ Загрузи динамическую память 1312 ⟩ Используется в разделе 1303.
⟨ Загрузи ещё пару вещей и завершающее контрольное слово 1327 ⟩ Используется в разделе 1303.
⟨ Загрузи константы для проверки целостности 1308 ⟩ Используется в разделе 1303.
⟨ Загрузи массив сведений о внутреннем шрифте номер k 1323 ⟩ Используется в разделе 1321.
⟨ Загрузи области с 1 по 6 массива *eqtb* 1317 ⟩ Используется в разделе 1314.
⟨ Загрузи пул строк 1310 ⟩ Используется в разделе 1303.
⟨ Загрузи сведения о шрифтах 1321 ⟩ Используется в разделе 1303.
⟨ Загрузи таблицу эквивалентов 1314 ⟩ Используется в разделе 1303.
⟨ Загрузи таблицы переносов 1325 ⟩ Используется в разделе 1303.
⟨ Загрузи хеш-таблицу 1319 ⟩ Используется в разделе 1314.
⟨ Заключительные процедуры 1333, 1335, 1336, 1338 ⟩ Используется в разделе 1330.
⟨ Закончи выдачу диагностического сообщения для переполненной или разреженной hbox 663 ⟩ Используется в разделе 649.
⟨ Закончи выдачу диагностического сообщения для переполненной или разреженной vbox 675 ⟩ Используется в разделе 668.
⟨ Закрой форматный файл 1329 ⟩ Используется в разделе 1302.
⟨ Замени z на z' и вычисли α, β 572 ⟩ Используется в разделе 571.
⟨ Замени буферную инструкцию на y или w и **goto** *found* 613 ⟩ Используется в разделе 612.
⟨ Замени буферную инструкцию на z или x и **goto** *found* 614 ⟩ Используется в разделе 612.
⟨ Замени остаток списка на p 1187 ⟩ Используется в разделе 1186.
⟨ Замени перенос по усмотрению на обязательный и установи $disc\_break \leftarrow true$ 882 ⟩ Используется в разделе 881.
⟨ Замени случай лексемы на p, если замена подходящая 1289 ⟩ Используется в разделе 1288.
⟨ Замени узлы $ha \,..\, hb$ последовательностью узлов, включающих переносы по усмотрению 903 ⟩ Используется в разделе 895.
⟨ Замени шрифт dvi_f на f 621 ⟩ Используется в разделе 620.
⟨ Замени этот узел на узел стиля, за которым следует подходящий выбор, затем **goto** *done_with_node* 731 ⟩ Используется в разделе 730.
⟨ Запиши новый годный разрыв 855 ⟩ Используется в разделе 851.
⟨ Запомни *cur_box* в регистре рамки 1077 ⟩ Используется в разделе 1075.
⟨ Запомни наибольшие значения в таблице *hyf* 924 ⟩ Используется в разделе 923.
⟨ Запрещённые случаи обнаружены в *main_control* 1048, 1098, 1111, 1144 ⟩ Используется в разделе 1045.
⟨ Запусти программу вывода пользователя и **return** 1025 ⟩ Используется в разделе 1012.
⟨ Иди к следующей строке файла или **goto** *restart*, если нет следующей строки, или **return**, если `\read` [читаемая] строка завершилась 360 ⟩ Используется в разделе 343.
⟨ Иди направо или выведи заполнитель 625 ⟩ Используется в разделе 622.
⟨ Извинись за незагруженный шрифт, **goto** *done* 567 ⟩ Используется в разделе 566.
⟨ Извинись за то, что нельзя выполнить действие, когда нет клея перед `\unskip` 1106 ⟩ Используется в разделе 1105.
⟨ Измени знаки всех трёх компонентов клея *cur_val* 431 ⟩ Используется в разделе 430.

⟨ Измени состояние, если нужно, и **goto** *switch*, если текущий символ следует пропустить, или **goto** *reswitch*, если текущий символ заменяется на другой 344 ⟩ Используется в разделе 343.
⟨ Измени спецификацию клея в *main_p* в соответствии с коэффициентом пробела 1044 ⟩ Используется в разделе 1043.
⟨ Измени текущий режим на $-vmode$ для `\halign`, и на $-hmode$ для `\valign` 775 ⟩ Используется в разделе 774.
⟨ Измени текущий стиль и **goto** *delete_q* 763 ⟩ Используется в разделе 761.
⟨ Или вставь материал, указанный в узле p, в подходящую рамку, или попридержи его до следующей страницы; также удали узел p из текущей страницы 1020 ⟩ Используется в разделе 1014.
⟨ Или добавь узел вставки p после узла q и удали его из текущей страницы, или удали $node(p)$ 1022 ⟩ Используется в разделе 1020.
⟨ Или обработай `\ifcase`, или установи b на значение булева условия 501 ⟩ Используется в разделе 498.
⟨ Используй код c, чтобы различать обобщённые дроби 1182 ⟩ Используется в разделе 1181.
⟨ Используй поля размера, чтобы разметить сведения о шрифте 566 ⟩ Используется в разделе 562.
⟨ Используй узел p, чтобы обновить текущие размеры высоты и глубины; если этот узел не суть допустимая точка разрыва, **goto** *not_found* или *update_heights*, иначе установи pi равным соответствующему штрафу на разрыве 973 ⟩ Используется в разделе 972.
⟨ Исправь высоту и глубину *cur_box*, для `\vtop` 1087 ⟩ Используется в разделе 1086.
⟨ Исправь счётчик ссылок, если нужно, и измени знак *cur_val*, если *negative* 430 ⟩ Используется в разделе 413.
⟨ Испытай шок от пропущенной левой фигурной скобки; **goto** *found* 475 ⟩ Используется в разделе 474.
⟨ Ищи в *eqtb* эквиваленты, равные p 255 ⟩ Используется в разделе 172.
⟨ Ищи в *hyph_list* указатели на p 933 ⟩ Используется в разделе 172.
⟨ Ищи в *save_stack* эквиваленты, указывающие на p 285 ⟩ Используется в разделе 172.
⟨ Ищи номер параметра или `##` 479 ⟩ Используется в разделе 477.
⟨ Ищи символы списка r в хеш-таблице, и установи *cur_cs* 374 ⟩ Используется в разделе 372.
⟨ Ищи слово $hc[1 \mathinner{..} hn]$ в таблице исключений и **goto** *found* (с *hyf*, содержащим переносы), если запись найдена 930 ⟩ Используется в разделе 923.
⟨ Константы во внешнем блоке 11 ⟩ Используется в разделе 4.
⟨ Копируй клей tabskip между колонками 795 ⟩ Используется в разделе 791.
⟨ Копируй список лексем 466 ⟩ Используется в разделе 465.
⟨ Копируй шаблоны из узла *cur_loop* в узел p 794 ⟩ Используется в разделе 793.
⟨ Локальные переменные для завершения выделенных формул 1198 ⟩ Используется в разделе 1194.
⟨ Локальные переменные для начальной настройки 19, 163, 927 ⟩ Используется в разделе 4.
⟨ Локальные переменные для переносов 901, 912, 922, 929 ⟩ Используется в разделе 895.
⟨ Локальные переменные для подсчётов форматирования 315 ⟩ Используется в разделе 311.
⟨ Локальные переменные для разрывов строк 862, 893 ⟩ Используется в разделе 815.
⟨ Локальные переменные для расчёта размеров 450 ⟩ Используется в разделе 448.
⟨ Метки во внешнем блоке 6 ⟩ Используется в разделе 4.
⟨ Найди активный узел с наименьшей дефектностью 874 ⟩ Используется в разделе 873.
⟨ Найди лучший активный узел для желаемой «свободности» 875 ⟩ Используется в разделе 873.
⟨ Найди лучший способ разбить вставку, и измени $type(r)$ на *split_up* 1010 ⟩ Используется в разделе 1008.
⟨ Найди места переноса для слова в *hc* или **return** 923 ⟩ Используется в разделе 895.
⟨ Найди наилучшие точки разрыва 863 ⟩ Используется в разделе 815.
⟨ Найди спецификацию клея *main_p* для текстовых пробелов в текущем шрифте 1042 ⟩ Используется в разделах 1041 и 1043.
⟨ Направь сообщение об ошибке, если $cur\_val = fmem\_ptr$ 579 ⟩ Используется в разделе 578.
⟨ Направь тело макроса и его параметров в считыватель 390 ⟩ Используется в разделе 389.
⟨ Настрой *selector* печати, основанный на *interaction* 75 ⟩ Используется в разделах 1265 и 1337.
⟨ Настрой всё доступное TEX'у 8 ⟩ Используется в разделе 4.
⟨ Настрой для переноса абзаца 891 ⟩ Используется в разделе 863.
⟨ Настрой заголовки особого списка и узлы констант 790, 797, 820, 981, 988 ⟩ Используется в разделе 164.

⟨ Настрой переменные с началом *ship_out* 617 ⟩ Используется в разделе 640.
⟨ Настрой процедуры ввода 331 ⟩ Используется в разделе 1337.
⟨ Настрой процедуры вывода 55, 61, 528, 533 ⟩ Используется в разделе 1332.
⟨ Настрой структуру hbox или vbox, затем **return** 1083 ⟩ Используется в разделе 1079.
⟨ Настрой таблицу (делает только INITEX) 164, 222, 228, 232, 240, 250, 258, 552, 946, 951, 1216, 1301, 1369 ⟩ Используется в разделе 8.
⟨ Начни или заверши ввод из файла 378 ⟩ Используется в разделе 367.
⟨ Начни новую текущую страницу 991 ⟩ Используется в разделах 215 и 1017.
⟨ Начни текущую страницу, вставь клей \topskip перед p и **goto** *continue* 1001 ⟩ Используется в разделе 1000.
⟨ Нумерованные случаи для *debug_help* 1339 ⟩ Используется в разделе 1338.
⟨ Обнови активные ширины, т. к. первый активный узел удалён 861 ⟩ Используется в разделе 860.
⟨ Обнови значение *printed_node* для символьного отображения 858 ⟩ Используется в разделе 829.
⟨ Обнови значения *first_mark* и *bot_mark* 1016 ⟩ Используется в разделе 1014.
⟨ Обнови значения *last_glue*, *last_penalty* и *last_kern* 996 ⟩ Используется в разделе 994.
⟨ Обнови значения max_h и max_v; но если страница слишком большая, **goto** *done* 641 ⟩ Используется в разделе 640.
⟨ Обнови измерения текущей страницы по отношению к клею или керну, указанному узлом p 1004 ⟩ Используется в разделе 997.
⟨ Обнови текущие измерения высоты и ширины по отношению к узлу клея или керна p 976 ⟩ Используется в разделе 972.
⟨ Обнови элемента ширины для объединённых колонок 798 ⟩ Используется в разделе 796.
⟨ Обнули $width(q)$ и следующий за этой колонкой клей tabskip 802 ⟩ Используется в разделе 801.
⟨ Обнули размеры 650 ⟩ Используется в разделах 649 и 668.
⟨ Обойдись без простых случаев пустых или плохих рамок 978 ⟩ Используется в разделе 977.
⟨ Обработай «нечто» p в цикле *vert_break*, **goto** *not_found* 1365 ⟩ Используется в разделе 973.
⟨ Обработай случаи, вызванные пробелами фигурными скобками, изменения состояния 347 ⟩ Используется в разделе 344.
⟨ Обработай узел или математический узел q насколько возможно, готовясь ко второму проходу *mlist_to_hlist*, затем перейди к следующему узлу в mlist 727 ⟩ Используется в разделе 726.
⟨ Обработай управляющую последовательность активного символа и установи $state \leftarrow mid\_line$ 353 ⟩ Используется в разделе 344.
⟨ Обрати ссылки соответствующих пассивных узлов, установив cur_p на первую точку разрыва 878 ⟩ Используется в разделе 877.
⟨ Обрезай текущий список, пока он не будет содержать только элементы *char_node*, *kern_node*, *hlist_node*, *vlist_node*, *rule_node* и *ligature_node*; установи в n длину списка, а в q — остаток списка 1121 ⟩ Используется в разделе 1119.
⟨ Общие переменные 13, 20, 26, 30, 32, 39, 50, 54, 73, 76, 79, 96, 104, 115, 116, 117, 118, 124, 165, 173, 181, 213, 246, 253, 256, 271, 286, 297, 301, 304, 305, 308, 309, 310, 333, 361, 382, 387, 388, 410, 438, 447, 480, 489, 493, 512, 513, 520, 527, 532, 539, 549, 550, 555, 592, 595, 605, 616, 646, 647, 661, 684, 719, 724, 764, 770, 814, 821, 823, 825, 828, 833, 839, 847, 872, 892, 900, 905, 907, 921, 926, 943, 947, 950, 971, 980, 982, 989, 1032, 1074, 1266, 1281, 1299, 1305, 1331, 1342, 1345 ⟩ Используется в разделе 4.
⟨ Объедини ширины в узлах объединения q с ширинами узлов p, не сохраняя узлы объединения q 803 ⟩ Используется в разделе 801.
⟨ Объяви подпроцедуры для *line_break* 826, 829, 877, 895, 942 ⟩ Используется в разделе 815.
⟨ Объяви подпроцедуры для *prefixed_command* 1215, 1229, 1236, 1243, 1244, 1245, 1246, 1247, 1257, 1265 ⟩ Используется в разделе 1211.
⟨ Объяви подпроцедуры для *var_delimiter* 709, 711, 712 ⟩ Используется в разделе 706.
⟨ Объяви процедуру *align_peek* 785 ⟩ Используется в разделе 800.
⟨ Объяви процедуру *fire_up* 1012 ⟩ Используется в разделе 994.
⟨ Объяви процедуру *get_preamble_token* 782 ⟩ Используется в разделе 774.
⟨ Объяви процедуру *handle_right_brace* 1068 ⟩ Используется в разделе 1030.

⟨ Объяви процедуру *init_span* 787 ⟩ Используется в разделе 786.
⟨ Объяви процедуру *insert_relax* 379 ⟩ Используется в разделе 366.
⟨ Объяви процедуру *macro_call* 389 ⟩ Используется в разделе 366.
⟨ Объяви процедуру *print_cmd_chr* 298 ⟩ Используется в разделе 252.
⟨ Объяви процедуру *print_skip_param* 225 ⟩ Используется в разделе 179.
⟨ Объяви процедуру *restore_trace* 284 ⟩ Используется в разделе 281.
⟨ Объяви процедуру *runaway* 306 ⟩ Используется в разделе 119.
⟨ Объяви процедуру *show_token_list* 292 ⟩ Используется в разделе 119.
⟨ Объяви процедуры для предобработки образцов переноса 944, 948, 949, 953, 957, 959, 960, 966 ⟩ Используется в разделе 942.
⟨ Объяви процедуры, используемые в *do_extension* 1349, 1350 ⟩ Используется в разделе 1348.
⟨ Объяви процедуры, используемые в *hlist_out*, *vlist_out* 1368, 1370, 1373 ⟩ Используется в разделе 619.
⟨ Объяви процедуры, нужные для показа элементов мат. списков (mlists) 691, 692, 694 ⟩ Используется в разделе 179.
⟨ Объяви процедуры, создающие математику 734, 735, 736, 737, 738, 743, 749, 752, 756, 762 ⟩ Используется в разделе 726.
⟨ Объяви процедуры, считывающие ограниченные классы целых 433, 434, 435, 436, 437 ⟩ Используется в разделе 409.
⟨ Объяви процедуры, считывающие сведения о шрифтах 577, 578 ⟩ Используется в разделе 409.
⟨ Объяви рабочие процедуры, используемые *main_control* 1043, 1047, 1049, 1050, 1051, 1054, 1060, 1061, 1064, 1069, 1070, 1075, 1079, 1084, 1086, 1091, 1093, 1095, 1096, 1099, 1101, 1103, 1105, 1110, 1113, 1117, 1119, 1123, 1127, 1129, 1131, 1135, 1136, 1138, 1142, 1151, 1155, 1159, 1160, 1163, 1165, 1172, 1174, 1176, 1181, 1191, 1194, 1200, 1211, 1270, 1275, 1279, 1288, 1293, 1302, 1348, 1376 ⟩ Используется в разделе 1030.
⟨ Объяви функцию *fin_mlist* 1184 ⟩ Используется в разделе 1174.
⟨ Объяви функцию *open_fmt_file* 524 ⟩ Используется в разделе 1303.
⟨ Объяви функцию *reconstitute* 906 ⟩ Используется в разделе 895.
⟨ Объясни, что слишком много мёртвых петель встретилось в строке 1024 ⟩ Используется в разделе 1012.
⟨ Оператор *case* для копирования различных типов и присваивания *words* числа ещё нескопированных начальных слов 206 ⟩ Используется в разделе 205.
⟨ Оправься от несбалансированной команды write 1372 ⟩ Используется в разделе 1371.
⟨ Оправься от несбалансированной программы вывода 1027 ⟩ Используется в разделе 1026.
⟨ Определи displacement d левой границы уравнения, по отношению к размеру строки z при условии, что $l = false$ 1202 ⟩ Используется в разделе 1199.
⟨ Определи значение $height(r)$ и подходящие установки клея; затем **return** или **goto** *common_ending* 672 ⟩ Используется в разделе 668.
⟨ Определи значение $width(r)$ и установки подходящего клея; затем **return** или **goto** *common_ending* 657 ⟩ Используется в разделе 649.
⟨ Определи порядок растяжимости 659 ⟩ Используется в разделах 658, 673 и 796.
⟨ Определи порядок сжимаемости 665 ⟩ Используется в разделах 664, 676 и 796.
⟨ Определи установки растяжимости вертикального клея, затем **return** или **goto** *common_ending* 673 ⟩ Используется в разделе 672.
⟨ Определи установки сжимаемости вертикального клея, затем **return** или **goto** *common_ending* 676 ⟩ Используется в разделе 672.
⟨ Определи установку растяжимости горизонтального клея, затем **return** или **goto** *common_ending* 658 ⟩ Используется в разделе 657.
⟨ Определи установку сжимаемости горизонтального клея, затем **return** или **goto** *common_ending* 664 ⟩ Используется в разделе 657.
⟨ Освободи место, занимаемое узлом p 999 ⟩ Используется в разделе 997.
⟨ Основные процедуры печати 57, 58, 59, 60, 62, 63, 64, 65, 262, 263, 518, 699, 1355 ⟩ Используется в разделе 4.
⟨ Отбрось ошибочные префиксы и **return** 1212 ⟩ Используется в разделе 1211.
⟨ Отбрось префиксы `\long` и `\outer`, если они неуместны 1213 ⟩ Используется в разделе 1211.
⟨ Отбрось текущую лексему и объясни, что она несогласована 1066 ⟩ Используется в разделе 1064.

⟨ Отдели часть вертикальной рамки, установи *cur_box* на неё 1082 ⟩ Используется в разделе 1079.
⟨ Открой *tfm_file* для ввода 563 ⟩ Используется в разделе 562.
⟨ Отправь рамку *p* 640 ⟩ Используется в разделе 638.
⟨ Передвинь *cur_p* к узлу, следующему за текущей строкой символов 867 ⟩ Используется в разделе 866.
⟨ Передвинь вперёд *r*; **goto** *found*, если ограничитель параметра полностью соответствует, иначе **goto** *continue* 394 ⟩ Используется в разделе 392.
⟨ Передвинь вперёд последний узел «нечто» в цикле *line_break* 1362 ⟩ Используется в разделе 866.
⟨ Передвинь вперёд последний узел «нечто» в цикле перед разбиением на строки 1363 ⟩ Используется в разделе 896.
⟨ Перейди к узлу *ha* или **goto** *done1*, если перенос не нужно делать 896 ⟩ Используется в разделе 894.
⟨ Перейди к узлу *hb*, вставляя буквы в *hu* и *hc* 897 ⟩ Используется в разделе 894.
⟨ Переключись на больший диакритический знак, если доступно и уместно 740 ⟩ Используется в разделе 738.
⟨ Перемести данные в *trie* 958 ⟩ Используется в разделе 966.
⟨ Перемести курсор за псевдолигатуру, затем **goto** *main_loop_lookahead* или *main_lig_loop* 1037 ⟩ Используется в разделе 1034.
⟨ Перемести символы узла лигатуры в *hu* и *hc*; но **goto** *done3*, если они не все суть буквы 898 ⟩ Используется в разделе 897.
⟨ Перемести список до разрыва 885 ⟩ Используется в разделе 882.
⟨ Перемести список после разрыва 884 ⟩ Используется в разделе 882.
⟨ Перемести узел *p* на текущую страницу; если пришло время разорвать страницу, помести узлы, следующие после разрыва обратно в список вклада, и **return** к программе вывода пользователя, если она есть 997 ⟩ Используется в разделе 994.
⟨ Перемести указатель *s* на конец текущего списка и установи *replace_count*(*r*) надлежащим образом 918 ⟩ Используется в разделе 914.
⟨ Перенеси узел *p* в настроечный (adjustment) список 655 ⟩ Используется в разделе 651.
⟨ Печатай баннер с датой и временем 536 ⟩ Используется в разделе 534.
⟨ Печатай вид списка лексем 314 ⟩ Используется в разделе 312.
⟨ Печатай две строки, используя ловкие псевдопечатанные сведения 317 ⟩ Используется в разделе 312.
⟨ Печатай или '`definition`', или '`use`', или '`preamble`', или '`text`' и вставь лексемы, которые должны привести к восстановлению 339 ⟩ Используется в разделе 338.
⟨ Печатай итог команды *c* 472 ⟩ Используется в разделе 470.
⟨ Печатай краткие сведения о содержимом узла *p* 175 ⟩ Используется в разделе 174.
⟨ Печатай недавно занятые позиции 171 ⟩ Используется в разделе 167.
⟨ Печатай перечень возможных действий 85 ⟩ Используется в разделе 84.
⟨ Печатай позицию текущей строки 313 ⟩ Используется в разделе 312.
⟨ Печатай символьное описание нового узла разрыва 846 ⟩ Используется в разделе 845.
⟨ Печатай символьное описание этого годного разрыва 856 ⟩ Используется в разделе 855.
⟨ Печатай список между *printed_node* и *cur_p*, затем установи *printed_node* ← *cur_p* 857 ⟩ Используется в разделе 856.
⟨ Печатай справку и **goto** *continue* 89 ⟩ Используется в разделе 84.
⟨ Печатай строку *s* как сообщение об ошибке 1283 ⟩ Используется в разделе 1279.
⟨ Печатай строку *s* на терминал 1280 ⟩ Используется в разделе 1279.
⟨ Печатать идентификатор шрифта для *font*(*p*) 267 ⟩ Используется в разделах 174 и 176.
⟨ Подави раскрытие следующей лексемы 369 ⟩ Используется в разделе 367.
⟨ Подай подходящий текст метки в сканер 386 ⟩ Используется в разделе 367.
⟨ Подай лексему *p* на вход TEX'а 326 ⟩ Используется в разделе 282.
⟨ Подготовь все рамки, участвующие во вставках действовать по очереди 1018 ⟩ Используется в разделе 1014.
⟨ Подправь конец строки, чтобы отразить вид разрыва, и включи `\rightskip`; также установи должное значение *disc_break* 881 ⟩ Используется в разделе 880.
⟨ Подстрой *shift_up* и *shift_down* для случая без дробной линии 745 ⟩ Используется в разделе 743.

⟨ Подстрой *shift_up* и *shift_down* для случая с дробной линией 746 ⟩ Используется в разделе 743.
⟨ Подстрой для коэффициента увеличения 457 ⟩ Используется в разделе 453.
⟨ Подстройся для установки `\globaldefs` 1214 ⟩ Используется в разделе 1211.
⟨ Пожалуйся на неизвестную единицу и **goto** *done2* 459 ⟩ Используется в разделе 458.
⟨ Пожалуйся на неопределённое семейство и установи *cur_i* пустым 723 ⟩ Используется в разделе 722.
⟨ Пожалуйся на неопределённый макрос 370 ⟩ Используется в разделе 367.
⟨ Пожалуйся на пропущенный `\endcsname` 373 ⟩ Используется в разделе 372.
⟨ Покажи вспомогательное поле, a 219 ⟩ Используется в разделе 218.
⟨ Покажи вставку p 188 ⟩ Используется в разделе 183.
⟨ Покажи заполнители p 190 ⟩ Используется в разделе 189.
⟨ Покажи значение переменной b 502 ⟩ Используется в разделе 498.
⟨ Покажи значение переменной *glue_set*(p) 186 ⟩ Используется в разделе 184.
⟨ Покажи идентификатор шрифта в *eqtb*$[n]$ 234 ⟩ Используется в разделе 233.
⟨ Покажи керн p 191 ⟩ Используется в разделе 183.
⟨ Покажи клей p 189 ⟩ Используется в разделе 183.
⟨ Покажи лексему (m, c) 294 ⟩ Используется в разделе 293.
⟨ Покажи лексему p и **return**, если есть трудности 293 ⟩ Используется в разделе 292.
⟨ Покажи лигатуру p 193 ⟩ Используется в разделе 183.
⟨ Покажи линейку p 187 ⟩ Используется в разделе 183.
⟨ Покажи мат. узел p 192 ⟩ Используется в разделе 183.
⟨ Покажи метку p 196 ⟩ Используется в разделе 183.
⟨ Покажи обычный математический узел p 696 ⟩ Используется в разделе 690.
⟨ Покажи особые поля неустановленного узла p 185 ⟩ Используется в разделе 184.
⟨ Покажи перенос по усмотрению p 195 ⟩ Используется в разделе 183.
⟨ Покажи полусловный код в *eqtb*$[n]$ 235 ⟩ Используется в разделе 233.
⟨ Покажи поправку p 197 ⟩ Используется в разделе 183.
⟨ Покажи рамку p 184 ⟩ Используется в разделе 183.
⟨ Покажи статус текущей страницы 986 ⟩ Используется в разделе 218.
⟨ Покажи стоимость разрыва вставки 1011 ⟩ Используется в разделе 1010.
⟨ Покажи текст раскрываемого макроса 401 ⟩ Используется в разделе 389.
⟨ Покажи текущее значение лексемы, затем **goto** *common_ending* 1294 ⟩ Используется в разделе 1293.
⟨ Покажи текущее значение некоторого параметра или регистра, затем **goto** *common_ending* 1297 ⟩ Используется в разделе 1293.
⟨ Покажи текущее содержимое рамки 1296 ⟩ Используется в разделе 1293.
⟨ Покажи текущий контекст 312 ⟩ Используется в разделе 311.
⟨ Покажи узел «нечто» p 1356 ⟩ Используется в разделе 183.
⟨ Покажи узел p 183 ⟩ Используется в разделе 182.
⟨ Покажи узел выбора p 695 ⟩ Используется в разделе 690.
⟨ Покажи узел дроби p 697 ⟩ Используется в разделе 690.
⟨ Покажи цену разрыва страницы 1006 ⟩ Используется в разделе 1005.
⟨ Покажи штраф p 194 ⟩ Используется в разделе 183.
⟨ Покажи эквивалент n в области 1 или 2 223 ⟩ Используется в разделе 252.
⟨ Покажи эквивалент n в области 3 229 ⟩ Используется в разделе 252.
⟨ Покажи эквивалент n в области 4 233 ⟩ Используется в разделе 252.
⟨ Покажи эквивалент n в области 5 242 ⟩ Используется в разделе 252.
⟨ Покажи эквивалент n в области 6 251 ⟩ Используется в разделе 252.
⟨ Положи символы *hu*$[l \mathinner{..} i]$ и знак переноса в *pre_break*(r) 915 ⟩ Используется в разделе 914.
⟨ Получи следующую лексему, которая не пуста и не знак; установи *negative* соответствующим образом 441 ⟩ Используется в разделах 440, 448 и 461.
⟨ Получи следующую непустую невызывающую лексему 406 ⟩ Используется в разделах 405, 441, 455, 503, 526, 577, 785, 791 и 1045.
⟨ Поменяй местами нижний и верхний индексы в рамке x 742 ⟩ Используется в разделе 738.

⟨ Помести в стек условий 495 ⟩ Используется в разделе 498.
⟨ Помести дробь в рамку с её ограничителями и установи *new_hlist*(*q*) на неё 748 ⟩
Используется в разделе 743.
⟨ Помести каждый примитив TEX'а в хеш-таблицу 226, 230, 238, 248, 265, 334, 376, 384, 411, 416, 468, 487, 491, 553, 780, 983, 1052, 1058, 1071, 1088, 1107, 1114, 1141, 1156, 1169, 1178, 1188, 1208, 1219, 1222, 1230, 1250, 1254, 1262, 1272, 1277, 1286, 1291, 1344 ⟩ Используется в разделе 1336.
⟨ Помести клей `\leftskip` слева и раздели эту строку 887 ⟩ Используется в разделе 880.
⟨ Помести клей `\rightskip` после узла *q* 886 ⟩ Используется в разделе 881.
⟨ Помести оптимальную текущую страницу в рамку 255, обнови *first_mark* и *bot_mark*, добавь вставки в их рамки, и помести оставшиеся (не использованные) узлы обратно в список вклада 1014 ⟩
Используется в разделе 1012.
⟨ Помести положительный размер 'at' в *s* 1259 ⟩ Используется в разделе 1258.
⟨ Помести символы *hu*[*i* + 1 . .] в *post_break*(*r*), добавляя к этому списку и к *major_tail*, пока не будет достигнута синхронизация 916 ⟩ Используется в разделе 914.
⟨ Помести справку в log-файл 90 ⟩ Используется в разделе 82.
⟨ Попробуй оправиться от несопряжённого `\right` 1192 ⟩ Используется в разделе 1191.
⟨ Попробуй перенести следующее слово 894 ⟩ Используется в разделе 866.
⟨ Попробуй получить другое имя log-файла 535 ⟩ Используется в разделе 534.
⟨ Попробуй последний разрыв строки в конце абзаца, и **goto** *done* если желаемые точки разрыва найдены 873 ⟩ Используется в разделе 863.
⟨ Попробуй разместить в узле *p* и в физически следующих за ним узлах и **goto** *found*, если размещение оказалось возможным 127 ⟩ Используется в разделе 125.
⟨ Попробуй разорвать после фрагмента по усмотрению, затем **goto** *done5* 869 ⟩
Используется в разделе 866.
⟨ Поругай недопустимый символ и **goto** *restart* 346 ⟩ Используется в разделе 344.
⟨ Посетуй, что `\the` не может сделать этого; верни нулевой результат 428 ⟩ Используется в разделе 413.
⟨ Построй рамку верхнего индекса *x* 758 ⟩ Используется в разделе 756.
⟨ Построй рамку нижнего индекса *x*, если нет верхнего индекса 757 ⟩ Используется в разделе 756.
⟨ Построй рамку с пределами выше и ниже неё, скошенными на *delta* 750 ⟩ Используется в разделе 749.
⟨ Преобразуй *cur_val* к более низкому уровню 429 ⟩ Используется в разделе 413.
⟨ Преобразуй *nucleus*(*q*) в горизонтальный список и присоедини верхний/нижний индексы 754 ⟩
Используется в разделе 728.
⟨ Преобразуй конечный *bin_noad* в *ord_noad* 729 ⟩ Используется в разделах 726 и 728.
⟨ Преобразуй математический клей к обычному 732 ⟩ Используется в разделе 730.
⟨ Прибери только что прочитанный параметр и проглоти его 400 ⟩ Используется в разделе 392.
⟨ Приведи в порядок память, удаляя узлы разрыва 865 ⟩ Используется в разделах 815 и 863.
⟨ Принеси внутренний размер и **goto** *attach_sign*, или принеси внутреннее целое 449 ⟩
Используется в разделе 448.
⟨ Приравняй клей к размеру 451 ⟩ Используется в разделах 449 и 455.
⟨ Присвоения 1217, 1218, 1221, 1224, 1225, 1226, 1228, 1232, 1234, 1235, 1241, 1242, 1248, 1252, 1253, 1256, 1264 ⟩
Используется в разделе 1211.
⟨ Присвой *b* значение негодности для растяжимости строки и вычисли соответствующий *fit_class* 852 ⟩
Используется в разделе 851.
⟨ Присвой *b* значение негодности для сжимаемости строки и вычисли соответствующий *fit_class* 853 ⟩
Используется в разделе 851.
⟨ Присвой значения *depth_threshold* ← *show_box_depth* и *breadth_max* ← *show_box_breadth* 236 ⟩
Используется в разделе 198.
⟨ Присоедини пределы к *y* и выровняй *height*(*v*), *depth*(*v*) с учётом их наличия 751 ⟩
Используется в разделе 750.
⟨ Присоедини список *p* к текущему списку и запиши его длину; затем заверши и **return** 1120 ⟩
Используется в разделе 1119.
⟨ Проверь значения «констант» на совместимость 14, 111, 290, 522, 1249 ⟩ Используется в разделе 1332.

⟨ Проверь контрольную сумму пула строк 53 ⟩ Используется в разделе 52.
⟨ Проверь на неправильное выравнивание в выделенной формуле 776 ⟩ Используется в разделе 774.
⟨ Проверь однословный список *avail* 168 ⟩ Используется в разделе 167.
⟨ Проверь список *avail* переменного размера 169 ⟩ Используется в разделе 167.
⟨ Проверь список символов на цикл 570 ⟩ Используется в разделе 569.
⟨ Проверь узел p в hlist, учитывая его влияние на размеры новой рамки, или перемещаю его в настроечный список; затем передвинь p к следующему узлу 651 ⟩ Используется в разделе 649.
⟨ Проверь узел p в vlist, учитывая его влияние на размеры новой рамки; затем передвинь p к следующему узлу 669 ⟩ Используется в разделе 668.
⟨ Проверь флаги недоступных узлов 170 ⟩ Используется в разделе 167.
⟨ Проверь, нечётное ли целое 504 ⟩ Используется в разделе 501.
⟨ Проверь, отношение между целыми или размерами 503 ⟩ Используется в разделе 501.
⟨ Проверь, следует ли другой `$` 1197 ⟩ Используется в разделах 1194, 1194 и 1206.
⟨ Проверь, совпадают ли два символа 506 ⟩ Используется в разделе 501.
⟨ Проверь, совпадают ли два текста макросов 508 ⟩ Используется в разделе 507.
⟨ Проверь, совпадают ли две лексемы 507 ⟩ Используется в разделе 501.
⟨ Проверь, состояние регистра рамки 505 ⟩ Используется в разделе 501.
⟨ Проверь, что $trie\_max \geqslant h + 256$ 954 ⟩ Используется в разделе 953.
⟨ Проверь, что необходимые шрифты для математических символов присутствуют; если нет, очисти текущие математические списки и присвой $danger \leftarrow true$ 1195 ⟩ Используется в разделах 1194 и 1194.
⟨ Проверь, что рамка 255 пуста перед началом вывода 1015 ⟩ Используется в разделе 1014.
⟨ Проверь, что рамка 255 пуста после вывода 1028 ⟩ Используется в разделе 1026.
⟨ Проверь, что узлы, следующие за hb, допускают перенос, и что по крайней мере $l\_hyf + r\_hyf$ букв найдено, иначе **goto** *done1* 899 ⟩ Используется в разделе 894.
⟨ Проверь, является ли узел p новой наиболее удачной точкой разрыва; затем, **goto** *done* если p — принудительный разрыв, или если страница уже достаточно полная 974 ⟩
Используется в разделе 972.
⟨ Проверь, является ли узел p новой наиболее удачной точкой разрыва; затем, если пришло время разорвать страницу, готовь для вывода, и, или запусти программу вывода пользователя и **return**, или выгрузи страницу и **goto** *done* 1005 ⟩ Используется в разделе 997.
⟨ Пройди второй раз по математическому списку, удаляя все математические узлы и вставляя подходящие пробелы и штрафы 760 ⟩ Используется в разделе 726.
⟨ Пройдись по списку преамбулы, определяя ширины колонок и заменяя записи выравнивания на временные неустановленные рамки 801 ⟩ Используется в разделе 800.
⟨ Пропусти до `\else` или `\fi`, затем **goto** *common_ending* 500 ⟩ Используется в разделе 498.
⟨ Пропусти операцию дроби и скажи о неясном случае 1183 ⟩ Используется в разделе 1181.
⟨ Просматривай варианты (z, x); установи f и c всякий раз, когда лучший символ найден; **goto** *found*, как только достаточно большой вариант встретился 707 ⟩ Используется в разделе 706.
⟨ Просматривай другие записи в стеке, пока не решишь, какую DVI-команду выдать; **goto** *found*, если узел p подходит 611 ⟩ Используется в разделе 607.
⟨ Просмотри все метки в узлах перед разрывом и установи последнюю ссылку на *null* в месте разрыва 979 ⟩ Используется в разделе 977.
⟨ Просмотри список символов, начиная с x в шрифте g; устанавливай f и c всякий раз, когда найден лучший символ; **goto** *found*, как только встретится достаточно большой вариант 708 ⟩
Используется в разделе 707.
⟨ Процедуры обработки ошибок 78, 81, 82, 93, 94, 95 ⟩ Используется в разделе 4.
⟨ Прочти и проверь данные шрифта; *abort* если TFM файл неправильный; если нет места для этого шрифта, скажи об этом и **goto** *done*; иначе *incr*(*font_ptr*) и **goto** *done* 562 ⟩
Используется в разделе 560.
⟨ Псевдопечатай список лексем 319 ⟩ Используется в разделе 312.
⟨ Псевдопечатай строку 318 ⟩ Используется в разделе 312.

⟨ Пусть *cur_h* — положение первой рамки, установи *leader_wd* + *lx* равным промежутку между соответствующими частями рамок 627 ⟩ Используется в разделе 626.
⟨ Пусть *cur_v* — положение первой рамки, установи *leader_wd* + *lx* равным промежутку между соответствующими частями рамок 636 ⟩ Используется в разделе 635.
⟨ Пусть *d* — естественная ширина этого клея; если растяжимый или сжимающийся, установи *v* ← *max_dimen*; **goto** *found* в случае заполнителей 1148 ⟩ Используется в разделе 1147.
⟨ Пусть *d* будет естественной шириной узла *p*; если узел «видимый», **goto** *found*; если узел — клей, который сжимается или растягивается, установи *v* ← *max_dimen* 1147 ⟩ Используется в разделе 1146.
⟨ Пусть *d* будет шириной «нечто» *p* 1361 ⟩ Используется в разделе 1147.
⟨ Пусть *n* — наибольшее допустимое значение кода, основанный на *cur_chr* 1233 ⟩ Используется в разделе 1232.
⟨ Разорви абзац в выбранных точках, выровняй полученные строки, чтобы исправить ширины и добавить их в текущий вертикальный список 876 ⟩ Используется в разделе 815.
⟨ Разорви текущую страницу на узле *p*, помести её в рамку 255 и помести оставшиеся узлы в список вклада 1017 ⟩ Используется в разделе 1014.
⟨ Раскрой макросы в список лексем и установи *link*(*def_ref*) на итог 1371 ⟩ Используется в разделе 1370.
⟨ Раскрой немакрос 367 ⟩ Используется в разделе 366.
⟨ Раскрой следующую часть ввода 478 ⟩ Используется в разделе 477.
⟨ Раскрой эту лексему после следующей 368 ⟩ Используется в разделе 367.
⟨ Рассматривай *cur_chr* как активный символ 1152 ⟩ Используется в разделах 1151 и 1155.
⟨ Рассмотри дефектность строки от *r* до *cur_p*; деактивируй узел *r*, если он не должен больше быть активным, затем **goto** *continue*, если строка от *r* до *cur_p* негодная, иначе запиши новый годный разрыв 851 ⟩ Используется в разделе 829.
⟨ Рассмотри узел с подходящей шириной; **goto** *found*, если он подходит 612 ⟩ Используется в разделе 611.
⟨ Рассчитай новую ширину строки 850 ⟩ Используется в разделе 835.
⟨ Расширь скользящие размеры в линейке *q* до границ выравнивания 806 ⟩ Используется в разделе 805.
⟨ Сделай волшебный расчёт 320 ⟩ Используется в разделе 292.
⟨ Сделай высоту рамки *y* равной *h* 739 ⟩ Используется в разделе 738.
⟨ Сделай неустановленный узел *r* в *hlist_node* ширины *w*, установив клей так, как если бы ширина была *t* 810 ⟩ Используется в разделе 808.
⟨ Сделай неустановленный узел *r* в *vlist_node* высоты *w*, установив клей так, как если бы высота была *t* 811 ⟩ Используется в разделе 808.
⟨ Сделай окончательные подстройки и **goto** *done* 576 ⟩ Используется в разделе 562.
⟨ Сделай список вклада пустым, установив его конец на *contrib_head* 995 ⟩ Используется в разделе 994.
⟨ Сделай узел *p* похожим на *char_node* и **goto** *reswitch* 652 ⟩ Используется в разделах 622, 651 и 1147.
⟨ Сделай узел лигатуры, если *ligature_present*; вставь пустой перенос по усмотрению, если уместно 1035 ⟩ Используется в разделе 1034.
⟨ Сделай частичную копию узла «нечто» *p* и установи *r* на него; присвой *words* число ещё нескопированных начальных слов 1357 ⟩ Используется в разделе 206.
⟨ Символ *k* не может печататься 49 ⟩ Используется в разделе 48.
⟨ Символ *s* суть текущий разделитель строк 244 ⟩ Используется в разделах 58 и 59.
⟨ Скажи пользователю, что что-то вышло из-под контроля и восстанови 338 ⟩ Используется в разделе 336.
⟨ Скопируй узел *p* в узел*r* 205 ⟩ Используется в разделе 204.
⟨ Случаи *flush_node_list*, возникающие только в математических списках 698 ⟩ Используется в разделе 202.
⟨ Случаи *handle_right_brace*, где *right_brace* запускает задержанное действие 1085, 1100, 1118, 1132, 1133, 1168, 1173, 1186 ⟩ Используется в разделе 1068.
⟨ Случаи *main_control*, которые не зависят от *mode* 1210, 1268, 1271, 1274, 1276, 1285, 1290 ⟩ Используется в разделе 1045.
⟨ Случаи *main_control*, которые не являются частью внутреннего цикла 1045 ⟩ Используется в разделе 1030.

⟨ Случаи *main_control*, которые служат для расширений TEX 1347 ⟩ Используется в разделе 1045.
⟨ Случаи *main_control*, которые строят рамки и списки 1056, 1057, 1063, 1067, 1073, 1090, 1092, 1094, 1097, 1102, 1104, 1109, 1112, 1116, 1122, 1126, 1130, 1134, 1137, 1140, 1150, 1154, 1158, 1162, 1164, 1167, 1171, 1175, 1180, 1190, 1193 ⟩ Используется в разделе 1045.
⟨ Случаи *print_cmd_chr* для символьной печати примитивов 227, 231, 239, 249, 266, 335, 377, 385, 412, 417, 469, 488, 492, 781, 984, 1053, 1059, 1072, 1089, 1108, 1115, 1143, 1157, 1170, 1179, 1189, 1209, 1220, 1223, 1231, 1251, 1255, 1261, 1263, 1273, 1278, 1287, 1292, 1295, 1346 ⟩ Используется в разделе 298.
⟨ Случаи *show_node_list*, которые возникают только в математических списках 690 ⟩ Используется в разделе 183.
⟨ Случаи для математических узлов, которые могут следовать за *bin_noad* 733 ⟩ Используется в разделе 728.
⟨ Случаи для узлов, которые могут появиться в mlist, после которых мы **goto** *done_with_node* 730 ⟩ Используется в разделе 728.
⟨ Случаи только для математики в нематематическом режиме, или наоборот 1046 ⟩ Используется в разделе 1045.
⟨ Случаи, когда символ пропускается 345 ⟩ Используется в разделе 344.
⟨ Смени уровень взаимодействия и **return** 86 ⟩ Используется в разделе 84.
⟨ Собери рамку x, объединяющую верхний и нижний индексы, со сдвинутым на *delta* верхним индексом 759 ⟩ Используется в разделе 756.
⟨ Собери рамку с vlist для дроби, согласно *shift_up* и *shift_down* 747 ⟩ Используется в разделе 743.
⟨ Собери расширяемый символ в новой рамке b, используя указания *rem_byte*(q) и шрифт f 713 ⟩ Используется в разделе 710.
⟨ Собирай константу, пока *cur_tok* подходящая цифра 445 ⟩ Используется в разделе 444.
⟨ Сожми уравнение настолько, насколько это возможно; если есть номер уравнения, который должен быть на отдельной строке, установи $e \leftarrow 0$ 1201 ⟩ Используется в разделе 1199.
⟨ Создай *format_ident*, открой форматный файл, и сообщи пользователю, что выгрузка началась 1328 ⟩ Используется в разделе 1302.
⟨ Создай активную точку разрыва, представляющую начало абзаца 864 ⟩ Используется в разделе 863.
⟨ Создай и добавь узел переноса по усмотрению, как альтернативу неперенесённому слову и продолжи развивать обе ветви, пока они не станут равноценны 914 ⟩ Используется в разделе 913.
⟨ Создай команду *down* или *right* для w и **return** 610 ⟩ Используется в разделе 607.
⟨ Создай команду *y0* или *z0*, чтобы повторно использовать предыдущее появление w 609 ⟩ Используется в разделе 607.
⟨ Создай новую спецификацию клея с шириной *cur_val*; считай составляющие растяжимости и сжимаемости 462 ⟩ Используется в разделе 461.
⟨ Создай новые активные узлы для лучших найденных годных разрывов 836 ⟩ Используется в разделе 835.
⟨ Создай первые 256 строк 48 ⟩ Используется в разделе 47.
⟨ Создай рамки x and z равной ширины для числителя и знаменателя, и вычисли значения по умолчанию *shift_up* и *shift_down*, на которые они будут отодвигаться от baseline 744 ⟩ Используется в разделе 743.
⟨ Создай узел символа p для *nucleus*(q), возможно, с последующим узлом керна для поправки на курсив, и присвой *delta* поправку на курсив, если есть нижний индекс 755 ⟩ Используется в разделе 754.
⟨ Создай узел символа q для следующего символа, но установи $q \leftarrow null$, если возникли проблемы 1124 ⟩ Используется в разделе 1123.
⟨ Создай узел страничной вставки с *subtype*$(r) = qi(n)$, и включи поправку клея для рамки n в состоянии текущей страницы 1009 ⟩ Используется в разделе 1008.
⟨ Сообщи о лишней правой скобке и **goto** *continue* 395 ⟩ Используется в разделе 392.
⟨ Сообщи о неверном выравнивании в выделенной формуле 1207 ⟩ Используется в разделе 1206.
⟨ Сообщи о негодном использовании макроса и прервись 398 ⟩ Используется в разделе 397.
⟨ Сообщи о переполнении входного буфера и прервись 35 ⟩ Используется в разделе 31.

⟨ Сообщи о переполненной вертикальной рамке vbox и **goto** *common_ending*, если эта рамка достаточно плоха 677 ⟩ Используется в разделе 676.
⟨ Сообщи о переполненной рамке hbox и **goto** *common_ending*, если эта рамка достаточно плоха 666 ⟩ Используется в разделе 664.
⟨ Сообщи о сбежавшем аргументе и прервись 396 ⟩ Используется в разделах 392 и 399.
⟨ Сообщи о слишком разреженной вертикальной рамке vbox и **goto** *common_ending*, если эта рамка достаточно плоха 674 ⟩ Используется в разделе 673.
⟨ Сообщи о слишком разреженной рамке hbox и **goto** *common_ending*, если эта рамка достаточно плоха 660 ⟩ Используется в разделе 658.
⟨ Сообщи о слишком тесной вертикальной рамке vbox и **goto** *common_ending*, если эта рамка достаточно плоха 678 ⟩ Используется в разделе 676.
⟨ Сообщи о слишком тесной рамке hbox и **goto** *common_ending*, если эта рамка достаточно плоха 667 ⟩ Используется в разделе 664.
⟨ Сообщи, что ошибочный код разделителя заменяется пустым; установи $cur\_val \leftarrow 0$ 1161 ⟩ Используется в разделе 1160.
⟨ Сообщи, что этот размер выходит за границы диапазона 460 ⟩ Используется в разделе 448.
⟨ Сообщи, что этот шрифт нельзя загрузить 561 ⟩ Используется в разделе 560.
⟨ Сооруди имя управляющей последовательности 372 ⟩ Используется в разделе 367.
⟨ Сохрани $save\_stack[save\_ptr]$ в $eqtb[p]$, если $eqtb[p]$ не содержит глобальное значение 283 ⟩ Используется в разделе 282.
⟨ Сохрани текущую лексему, но **goto** *continue*, если она — пробел, который мог бы стать объединяющим параметром 393 ⟩ Используется в разделе 392.
⟨ Считай fil-единицы; **goto** *attach_fraction* если найдена 454 ⟩ Используется в разделе 453.
⟨ Считай mu-единицы и **goto** *attach_fraction* 456 ⟩ Используется в разделе 453.
⟨ Считай аргумент для команды c 471 ⟩ Используется в разделе 470.
⟨ Считай десятичную дробь 452 ⟩ Используется в разделе 448.
⟨ Считай и построй параметрическую часть определения макроса 474 ⟩ Используется в разделе 473.
⟨ Считай и построй тело списка лексем; **goto** *found* когда закончишь 477 ⟩ Используется в разделе 473.
⟨ Считай имя файла в буфере 531 ⟩ Используется в разделе 530.
⟨ Считай код алфавитного символа в *cur_val* 442 ⟩ Используется в разделе 440.
⟨ Считай необязательный пробел 443 ⟩ Используется в разделах 442, 448, 455 и 1200.
⟨ Считай параметры и установи $link(r)$ на тело макроса; но **return**, если обнаружена недопустимая команда `\par` 391 ⟩ Используется в разделе 389.
⟨ Считай подформулу, заключённую в фигурных скобках и **return** 1153 ⟩ Используется в разделе 1151.
⟨ Считай преамбулу и запиши её в список *preamble* 777 ⟩ Используется в разделе 774.
⟨ Считай спецификацию размера шрифта 1258 ⟩ Используется в разделе 1257.
⟨ Считай управляющую последовательность и установи $state \leftarrow skip\_blanks$ или *mid_line* 354 ⟩ Используется в разделе 344.
⟨ Считай числовую константу 444 ⟩ Используется в разделе 440.
⟨ Считай шаблон $\langle u_j \rangle$, помещая итоговый список лексем в *hold_head* 783 ⟩ Используется в разделе 779.
⟨ Считай шаблон $\langle v_j \rangle$, помещая итоговый список лексем в *hold_head* 784 ⟩ Используется в разделе 779.
⟨ Считывай все другие единицы и установи *cur_val* и f соответствующим образом; **goto** *done* в случае масштабных пунктов 458 ⟩ Используется в разделе 453.
⟨ Считывай единицы внутренних размеров; **goto** *attach_sign* с установленным *cur_val*, если найдено 455 ⟩ Используется в разделе 453.
⟨ Считывай единицы и установи *cur_val* равным $x \cdot (cur\_val + f/2^{16})$, где x есть sp на единицу; **goto** *attach_sign*, если единицы внутренние 453 ⟩ Используется в разделе 448.
⟨ Считывай параметр, пока не будет найдена его строка-ограничитель; или, если $s = null$, просто считай строку-ограничитель 392 ⟩ Используется в разделе 391.
⟨ Считывай текст преамбулы, пока *cur_cmd* не есть *tab_mark* или *car_ret*, ища изменения в клее tabskip; добавь запись выравнивания к списку преамбулы 779 ⟩ Используется в разделе 777.
⟨ Текущий *mem* эквивалент параметра клея номер n 224 ⟩ Используется в разделах 152 и 154.

⟨ Типы во внешнем блоке 18, 25, 38, 101, 109, 113, 150, 212, 269, 300, 548, 594, 920, 925 ⟩ Используется в разделе 4.
⟨ Убедись, что *page_max_depth* не превышено 1003 ⟩ Используется в разделе 997.
⟨ Убедись, что *pi* в верных пределах 831 ⟩ Используется в разделе 829.
⟨ Убери нежелательные узлы в начале следующей строки 879 ⟩ Используется в разделе 877.
⟨ Увеличь ещё память переменного размера и **goto** *restart* 126 ⟩ Используется в разделе 125.
⟨ Увеличь число параметров в последнем шрифте 580 ⟩ Используется в разделе 578.
⟨ Удали лексемы $c - \texttt{"0"}$ и **goto** *continue* 88 ⟩ Используется в разделе 84.
⟨ Удали последнюю рамку, если она не является частью переноса по усмотрению 1081 ⟩
Используется в разделе 1080.
⟨ Удали узлы страниц-вставок 1019 ⟩ Используется в разделе 1014.
⟨ Удивись, что здесь нет числа 446 ⟩ Используется в разделе 444.
⟨ Удлиняй периодически преамбулу 793 ⟩ Используется в разделе 792.
⟨ Ужаснись от того, что выравнивание не сейчас обрабатывается 1128 ⟩ Используется в разделе 1127.
⟨ Укажи, что пользователю следует сказать `\mathaccent` 1166 ⟩ Используется в разделе 1165.
⟨ Уничтожь t узлов, следующих за q, и установи r на следующий узел 883 ⟩ Используется в разделе 882.
⟨ Уничтожь узел «нечто» p и **goto** *done* 1358 ⟩ Используется в разделе 202.
⟨ Упакуй неустановленную рамку для текущей колонки и запиши её ширину 796 ⟩
Используется в разделе 791.
⟨ Упакуй семейство в *trie* относительно h 956 ⟩ Используется в разделе 953.
⟨ Упакуй список преамбулы, чтобы определить действительные величины клея tabskip, и пусть p указывает на эту рамку прототипа 804 ⟩ Используется в разделе 800.
⟨ Упорядочь p в списке, начиная с *rover* и перемести p на *rlink*(p) 132 ⟩ Используется в разделе 131.
⟨ Упорядочь таблицы операций переносов в надлежащем порядке 945 ⟩ Используется в разделе 952.
⟨ Упрости тривиальную рамку 721 ⟩ Используется в разделе 720.
⟨ Установи значение *output_penalty* 1013 ⟩ Используется в разделе 1012.
⟨ Установи значения *cur_size* и *cur_mu* на основе *cur_style* 703 ⟩
Используется в разделах 720, 726, 730, 754, 760 и 763.
⟨ Установи клей в узле r и измени его тип с неустановленного узла 808 ⟩ Используется в разделе 807.
⟨ Установи клей во всех неустановленных рамках текущего списка 805 ⟩ Используется в разделе 800.
⟨ Установи начальные значения ключевых переменных 21, 23, 24, 74, 77, 80, 97, 166, 215, 254, 257, 272, 287, 383, 439, 481, 490, 521, 551, 556, 593, 596, 606, 648, 662, 685, 771, 928, 990, 1033, 1267, 1282, 1300, 1343 ⟩
Используется в разделе 8.
⟨ Установи неустановленную рамку q и неустановленные рамки в ней 807 ⟩ Используется в разделе 805.
⟨ Установи параметры длины строки, готовясь к подвешенному отступу 849 ⟩
Используется в разделе 848.
⟨ Установи переменную b на рамку для (f, c) 710 ⟩ Используется в разделе 706.
⟨ Установи переменную c на текущий символ начала управляющей последовательности 243 ⟩
Используется в разделе 63.
⟨ Установи структуры данных с курсором, следующим за позицией j 908 ⟩ Используется в разделе 906.
⟨ Читай в буфере вперёд пока не найдёшь небукву; если встретился раскрываемый код, упрости его и **goto** *start_cs*; иначе, если многобуквенная управляющая последовательность найдена, настрой *cur_cs* и *loc*, и **goto** *found* 356 ⟩ Используется в разделе 354.
⟨ Читай данные символов 569 ⟩ Используется в разделе 562.
⟨ Читай заголовок `TFM` 568 ⟩ Используется в разделе 562.
⟨ Читай одну строку, но верни *false*, если место в памяти строк слишком мало 52 ⟩
Используется в разделе 51.
⟨ Читай остальные строки из файла `TEX.POOL` и верни *true*, или дай сообщение об ошибке и верни *false* 51 ⟩ Используется в разделе 47.
⟨ Читай параметры шрифта 575 ⟩ Используется в разделе 562.
⟨ Читай первую строку нового файла 538 ⟩ Используется в разделе 537.
⟨ Читай поля размера `TFM` 565 ⟩ Используется в разделе 562.
⟨ Читай программу лигатур и кернов 573 ⟩ Используется в разделе 562.

⟨ Читай размеры рамок 571 ⟩ Используется в разделе 562.
⟨ Читай рецепты расширяемых символов 574 ⟩ Используется в разделе 562.
⟨ Читай следующую строку из файла в *buffer* или **goto** *restart*, если файл закончился 362 ⟩
Используется в разделе 360.