Aufgabe 1: `\filename`

Ersetzen von Tokens

Schreibt ein Kommando `\filename`, das den Namen einer Datei mit `\emph` oder `\texttt` formatiert – je nach dem, wie es euch besser gefällt. In dem Dateinamen sollen Unterstriche ohne Escaping verwendet werden können.

Bonus

Haltet das Ersetzen der Unterstriche expandierbar.

Beispiel

```
\filename{__init__.py}
```
anstatt
```
\emph{\_\_init\_\_.py}
```

Hinweis

Um das Kommando auch in Argumenten zu anderen Kommandos verwenden zu können (wie z. B. in `\caption`), sollte es *nicht* auf dem Ändern von Catcodes basieren.

Schreibt für das Ersetzen der Unterstriche ein separates Kommando, das in `\filename` verwendet wird und auch in anderen Situationen verwendet werden kann.

Hilfe

Mit Hilfe von Parameter-Delimiters kann eine Liste von Tokens in zwei Teile aufgesplitted werden, den Teil *vor* dem ersten Unterstrich und den Rest *hinter* dem ersten Unterstrich.

Überlegt euch einen sinnvollen Parameter-Delimiter, mit dem ihr das Ende des Dateinamens (bzw. der Liste an Tokens, in denen die Unterstriche ersetzt werden sollen) markieren könnt.

Ein möglicher Ansatz wäre: `\def\@repl@underscores#1_#2\relax{%`

Wie muss das Kommando aufgerufen werden muss, damit es auch dann, wenn kein Unterstrich vorkommt, nicht zu einem `Paragraph/File ended before \command was complete` Fehler kommt?

Wenn `#2` leer ist kam kein (weiterer) Unterstrich vor und `#1` ist das Ergebnis.

Wenn `#2` *nicht* leer ist, kam mindestens ein Unterstrich vor. Gebt `#1` aus, fügt `\textunderscore` anstatt des Unterstriches ein und führt das Kommando nochmal mit `#2` aus um potentiell weiter vorhandene Unterstriche zu ersetzen.

Bedenkt, dass `#2` mit dem zusätzlich angefügten Unterstrich endet, daher muss dieser nicht noch einmal angehängt werden.

Aufgabe 2: `\strippath`

Löschen von Tokens

[Diese Aufgabe ist sehr ähnlich zu Aufgabe 1. Wenn ihr Aufgabe 1 bereits gelöst habt, ist eine andere Aufgabe vielleicht interessanter.]

Schreibt ein Kommando `\strippath`, das den kompletten Namen einer Datei bekommt und den Pfad abschneidet, sodass nur der Name der Datei überbleibt.

Beispiel

```
\newcommand{\fullfilename}{path/to/a/file.tex}
\strippath\fullfilename
```

soll „file.tex“ ausgeben.

Bonus

Haltet das Kommando expandierbar.

Hinweis

Auch auf Windows soll der Separator ein Slash sein (kein Backslash), so wie bei `\input` oder `\includegraphics`.

Hilfe

Versucht erstmal die Aufgabe zu lösen, wenn der Dateipfad direkt übergeben wird und nicht in einem Kommando.

Beginnt damit ein Kommando zu schreiben, das den Dateinamen am ersten Slash aufsplitted.

Achtet darauf, dass ihr keinen `Paragraph/File ended before \command was complete` Fehler bekommt, unabhängig davon, ob der Pfad einen Slash enthält oder nicht. Überlegt euch, wie die Parameterdelimiter und der Aufruf dieses Kommandos dafür aussehen müssen.

Anhand des zweiten Argumentes (das nach dem Slash) kann überprüft werden, ob der Dateipfad einen Slash enthielt.

Wenn der Dateipfad *keinen* Slash enthielt, ist `#1` die Lösung.

Wenn der Dateipfad einen Slash enthielt, kann `#1` verworfen werden und das Kommando auf `#2` erneut angewendet werden.

Schreibt einen Wrapper für das Kommando, der die geforderte Syntax erfüllt. Mit `\expandafter` könnt ihr das Argument vor dem Kommando expandieren.

Aufgabe 3: Clevere Referenzen

Mehrere Versionen

Schreibt einen Wrapper um `\cref`, der zusätzlich die Seitenzahl ausgibt, wenn sich das Label auf einer anderen Seite befindet als die Referenz.

Wie könnt ihr einen zweiten, entsprechenden Wrapper um `\Cref` mit möglichst wenig Code-Duplizierung definieren?

Hintergrund

Mit dem `\label`-Kommando können in einem Dokument bestimmte Stellen, wie z. B. Abbildungen, Tabellen oder Abschnitte gespeichert werden. Danach, aber auch schon davor, können diese Labels mit `\ref` und `\pageref` referenziert werden. `\ref` gibt die (Abbildungs-/Tabellen-/Abschnitts-)Nummer an und `\pageref` die entsprechende Seitenzahl.

Das `cleveref`-Paket definiert das `\cref`-Kommando, das im Gegensatz zu `\ref` angibt, auf welche Art von Objekt referenziert wird (Abbildung/Tabelle/Abschnitt). Dies bietet die Vorteile von weniger Schreibaufwand, einheitlicher Benennung (nicht mal *image*, mal *graphic*, mal *picture*) und dass man nicht vergessen kann ein geschütztes Leerzeichen (`~`) zwischen Benennung und Nummer zu verwenden. Mit der `nameinlink` Paketoption wird auch die Benennung Teil des Links, wenn das `hyperref`-Paket geladen ist. (`cleveref` ist eine der Ausnahmen, die nach `hyperref` geladen werden muss, das sonst für gewöhnlich als letztes geladen werden muss.)

Des Weiteren bietet `\cref` den Vorteil, dass man mehrere Labels, über Kommas getrennt, gleichzeitig referenzieren kann.

Es wird unterschieden zwischen den beiden Kommandos `\cref`, das innerhalb eines Satzes verwendet wird, und `\Cref`, das am Anfang eines Satzes verwendet wird. In manchen Sprachen, wie z. B. Englisch, ist diese Unterscheidung wichtig, weil am Satzanfang groß geschrieben wird und innerhalb des Satzes klein. Im Deutschen ist `cleveref` standardmäßig so konfiguriert, den Begriff *Abbildung* am Satzanfang auszuschreiben und innerhalb des Satzes abzukürzen. Wem diese Einstellung nicht gefällt, kann die Paketoption `noabbrev` verwenden.

Seitenzahlen muss man normalerweise aber trotzdem noch per Hand einfügen. Wenn sich Label und Referenz auf der selben Doppelseite befinden ist eine Seitenangabe überflüssig, weil der Blick ohnehin auf das referenzierte Objekt fällt, noch bevor der Leser den Verweis liest. Im Gegenteil, eine Seitenangabe würde den Leser verunsichern, sodass er nochmal die Seitenzahlen prüfen müsste, und würde unnötiges Rauschen in den Text einfügen, der vom eigentlichen Inhalt ablenken würde. Wenn sich das referenzierte Objekt jedoch auf einer anderen Doppelseite befindet, ist es wichtig eine Seitenzahl anzugeben, damit der Leser nicht suchen muss. Oft weiß man beim Tippen aber nicht, ob sich Label und Referenz auf der selben Doppelseite befinden werden und selbst wenn, kann sich das mit dem Einfügen eines Absatzes oder einer Abbildung ganz schnell ändern. Deshalb wäre es praktisch, wenn das Einfügen von Seitenzahlen automatisch passiert – von dem gesparten Schreibaufwand mal ganz abgesehen.

Hinweis

Der LaTeX-Counter `page` beinhaltet die aktuelle Seitenzahl.

Normalerweise verwendet man `\the`*`countername`* um auf den Wert eines LaTeX-Counters zuzugreifen. Für diesen Vergleich brauchen wir aber in jedem Fall den Zahlwert, deshalb ist es sicherer `\arabic{`*`countername`*`}` zu verwenden, für den Fall, dass `\thepage` umdefiniert worden ist, sodass es zu einer nicht-arabischen Zahl oder einer Zahl mit einem Affix expandiert.

Es ist jedoch unvermeidlich, dass der Zugriff auf den Counter expandiert wird, bevor ein eventueller Seitenumbruch festgelegt wird. Das bedeutet, dass `\arabic{page}` zu einem falschen Wert expandiert, wenn vor diesem `\arabic{page}` innerhalb desselben Absatzes oder ggf. zwischen diesem Absatz und dem vorherigen Absatz (ohne ein explizites `\newpage`) ein Seitenumbruch auftritt.

`\pageref` statt `\thepage` zu verwenden würde in mehr Fällen das richtige Ergebnis bringen. Jedoch würde das deutlich mehr Overhead bedeuten. Für jede Referenz müsste ein Label angelegt werden und man bräuchte einen Counter um für jede Referenz ein eindeutiges Label zu generieren. Wenn man eine weitere Referenz zwischendurch einfügen würde, würden sich alle Referenzen verschieben und man müsste das Dokument einmal mehr kompilieren. Aber auch mit diesem Ansatz könnte wahrscheinlich nicht ausgeschlossen werden, dass stellenweise eine Seitenangabe eingefügt werden würde wo man lieber keine hätte oder andersherum, wenn die Referenz auf einen Seitenumbruch fällt. Im Worst Case wäre vielleicht sogar nicht konvergierendes Verhalten möglich.

Bonus 1

Ergänzt die beiden Wrapper um eine Sternchen Version, die eine Seitenzahl erzwingt und eine weitere Version, die durch ein Suffix eurer Wahl getriggert wird und die eine Seitenzahl unterbindet.

Die zweite Version dieses Kommandos hat dem normalen `\cref` gegenüber den Vorteil des optionalen Argumentes, das im nächsten Bonus implementiert wird.

Das `suffix`-Paket kann Schreibarbeit ersparen um starred und andere Versionen eines Kommandos zu definieren.

Bonus 2

In der deutschen Sprache wird zwar viel über Partikel geregelt, aber im Genitiv Singular maskuliner und neutraler Begriffe ändert sich das Wort selber. (Dies kommt eher selten vor, auch deshalb weil die Standard-LaTeX Floating-Umgebungen feminin sind, aber mit dem `float`-Paket können weitere Floating-Umgebungen definiert werden und es ist besser gerüstet zu sein.) Erweitert daher `\cpref` & Freunde um ein optionales Argument, das, wenn es gegeben ist, den Objekttyp ersetzt. z. B.:

```
Die Bedeutung des \cpref[Graphens]{sin2}...
```

Bonus 3

Wenn viele Referenzen auf einmal vorkommen, beispielsweise am Anfang eines Abschnitts: „In Unterabschnitt 1.1 …, in Unterabschnitt 1.2 …, …“ stören die Seitenzahlen mehr als das sie helfen. Definiert daher ein Kommando, das bis zum Ende der aktuellen Gruppe alle Referenzen ohne Seitenzahl setzt.

Und wenn ihr schon dabei seid, könnt ihr auch gleich ein gegenteiliges anlegen, das bei allen Referenzen Seitenzahlen setzt.

Bonus 4

In einem doppelseitig gesetzten Dokument stellt sich nicht die Frage, ob sich eine Referenz auf der selben Seite befindet, sondern ob sie sich auf der selben *Doppelseite* befindet. Ändert das Kommando dementsprechend.

Bonus 5

Das `\cref`-Kommando kann mit mehreren Labels gleichzeitig umgehen (kommaseparierte Liste). Erweitert euer Kommando so, dass es das auch kann.

Hilfe

Expandierbarkeit ist nicht relevant, da `\cref` ohnehin nicht expandierbar ist.

`\cpageref` ist nicht expandierbar, daher könnt ihr es nicht für den Vergleich verwenden.

Alternativ kann man `\pageref` verwenden. Dieses unterstützt zwar nur eine Referenz, aber darum könnt ihr euch später kümmern.

Die Expansion von `\pageref` enthält eine `\hbox{}`, weshalb der folgende Vergleich fehl schlägt:

```
\edef\@cpref@thispage{\arabic{page}}%
\edef\@cpref@referencepage{\pageref{#1}}%
\ifx \@cpref@thispage \@cpref@referencepage
```

Dies lässt sich umgehen, in dem man lokal `\let\hbox=\@firstofone` einsetzt oder stattdessen `\ifthenelse{\equal{...}{...}}` aus dem `ifthen`-Paket verwendet.

`\@for\thislabel:=#1\do{...}` kann verwendet werden um über eine kommaseparierte Liste zu iterieren.

Aufgabe 4: \verbarg

Ändern von Catcodes

Schreibt ein Kommando `\verbarg`, das es einfach machen soll, Kommandos zu schreiben, die ein `\verb`-ähnliches Argument nehmen.

`\verbarg` soll zwei Argumente nehmen: Das erste ist ein Kommando, das einen Undelimited Parameter hat. Das zweite ist ein Argument, das wie bei `\verb` beliebige Zeichen enthalten kann und durch das gleiche Zeichen beendet wird, mit dem es geöffnet wird.

Beispiel

```
\verbarg\texttt|^_^|
```

soll äquivalent sein zu

```
\verb|^_^|
```

Bonus 1

Im Gegensatz zu `\verb` soll das Argument, wenn es mit `(` eingeleitet wird, mit `)` beendet werden. Entsprechend für `[` und `{`.

Wenn das Argument mit einem Leerzeichen eingeleitet wird, soll es bis zum Ende der Zeile gehen. Wenn an Stelle des Argumentes ein Zeilenumbruch steht, soll das Argument leer bleiben.

Bonus 2

`beamer`-Klasse: Benutzt `\verbarg` um das `\verb`-Kommando overlayaware zu machen. (s. Hintergrund zu Aufgabe 5)

Ligaturen, wie das Verschmelzen von zwei Bindestrichen zu einem Gedankenstrich, sollen dabei nicht auftreten.

Hinweis

Die LaTeX Kommandos `\dospecials` und `\@makeother` sind dafür nützlich. Schaut euch deren Definition in *source2e* an.

Das `\appto`-Kommando aus dem `etoolbox`-Paket erspart Schreibarbeit beim Anhängen von Tokens an den Replacementtext eines Kommandos.

Das `\verb`-Kommando verwendet das LaTeX Kommando `\@noligs` um (manche) Ligaturen zu deaktivieren, damit ein doppelter Bindestrich nicht zu einem Gedankenstrich verschmilzt. Bei der Verwendung von `\@noligs` müssen zwei Dinge beachtet werden: (1) es basiert auf dem Ändern von Catcodes, macht

die entsprechenden Zeichen aktiv, und (2) die Definition der aktiven Zeichen ist fragil.

In manchen Anwendungen ist `\@noligs` unerwünscht, beispielsweise wenn das Kommando-Argument als Kommando-Zeilen-Argument für den Aufruf eines externen Programmes dienen soll. Daher würde ich es nicht in `\verbarg` integrieren. `\verbarg` sollte aber so programmiert werden, dass es mit `\@noligs` kompatibel ist (d. h. dass es mit fragilen, aktiven Zeichen im Argument umgehen kann.)

(Ohne `fontenc` T1 ist `\@noligs` für die Standard-Typewriter-Schrift nicht erforderlich, da diese Schrift keine Ligaturen für Gedankenstriche zu enthalten scheint. `fontenc` T1 sollte aber in allen (deutschsprachigen) Dokumenten verwendet werden, damit Umlaute als einzelnes Zeichen dargestellt werden können. Andernfalls gibt es Probleme, wenn man Umlaute aus dem pdf herauskopieren möchte. Außerdem funktioniert ohne T1 keine automatische Silbentrennung in Wörtern mit Umlauten. Man sollte aber beachten, dass die Verwendung von T1 in Abhängigkeit der auf dem System installierten Schriften zu einer unsauberen, verpixelten Schrift führen kann, die beim Kopieren von Ligaturen Probleme bereiten kann. Abhilfe schafft das der Standardschrift sehr ähnliche Paket `lmodern`.)

Hilfe

Öffnet eine Gruppe. Speichert das Kommando, das als erstes Argument gegeben wird, in einem Kommando ab.

Ändert die Catcodes aller Zeichen auf Other mit Hilfe von `\dospecials` und `\@makeother`.

Speichert das nächste Token in einem Kommando und definiert ein leeres Kommando, in dem ihr die folgenden Tokens speichert.

Iteriert über die folgenden Tokens, bis ein Token von gleichem Character Code wie das gespeicherte eingelesen wird. Hängt alle dazwischenliegenden Tokens an das dafür vorgesehene Kommando an und verwerft das End-Token.

Zum Iterieren braucht kein `\futurelet` verwendet zu werden – das jeweils nächste Token als undelimited Argument einzulesen funktioniert, weil alle Tokens Catcode Other haben (oder Catcode Active mit `\@noligs`).

Macht es einen Unterschied ob man `\ifx` oder `\if` verwendet?

Im Normalfall nicht, weil sich die Catcodes während des Einlesen des Argumentes nicht ändern. Jedoch muss man bei der Verwendung von `\if` bedenken `\noexpand` zu verwenden, weil mit `\@noligs` neben Catcode Other auch Catcode Active auftreten kann. (Bei `\ifx` ist das nicht erforderlich, weil `\ifx` im Gegensatz zu `\if` nicht expandiert.)

Wenn aber das Wrapper-Kommando, in dem `\verbarg` enthalten ist, als letztes Argument, bevor `\verbarg` in Aktion tritt, ein optionales Argument akzeptiert (beispielsweise eine Overlayspecification in der `beamer`-Klasse) *und* dieses optionale Argument *nicht* übergegeben wird, dann wird der öffnende Argument-

Delimiter des `\verbarg`-Argumentes gelesen, noch bevor die Catcodes geändert worden sind. Dann können, wenn man `\ifx` verwendet, nur Zeichen, deren Catcode nicht geändert wird – also alle Zeichen der Kategorie Other, die nicht von `\@noligs` auf Active geändert werden – als Argument-Delimiter verwendet werden. Wenn man hingegen `\if` verwendet, dann können deutlich mehr Zeichen verwendet werden, aber nicht alle. Beispielsweise Zeichen, die für gewöhnlich der Kategorie Comment angehören, können prinzipiell nicht verwendet werden, wenn das optionale Argument *nicht* gegeben ist. Welche weitere Zeichen das betrifft ist implementierungsabhängig.

Das `\expandonce`-Kommando aus dem `etoolbox`-Paket kann *nicht* verwendet werden um die Expandierung eines `\if`s zu Kontrollieren, weil es auf dem ε-TEX-Primitiv `\unexpanded` basiert. Verwendet stattdessen `\expandafter \noexpand`. s. `https://tex.stackexchange.com/a/497646/120953`

Für das Anhängen von Tokens an ein Kommando gibt es verschiedene Möglichkeiten:

```
\expandafter \def \expandafter \arg \expandafter {\arg #1} % TEX
\edef\arg{\unexpanded\expandafter{\arg #1}} % ε-TEX
\appto\arg{#1} % \usepackage{etoolbox}
```

siehe auch `https://tex.stackexchange.com/q/342506/120953`

Expandiert das Kommando, in dem ihr das Argument gesammelt habt, genau ein mal und hängt es in geschweiften Klammern (Catcodes 1 und 2) hinter das auszuführende Kommando.

Bonus: Am besten schließt ihr die Gruppe noch bevor das Kommando ausgeführt wird, damit das Kommando hinter dem gespeicherten verbarg noch weitere Kommandos einlesen kann, ohne über das `\endgroup` zu stolpern. Natürlich müssen die Kommandos, in denen das auszuführende Kommando und das Argument gespeichert sind, vor dem Schließen der Gruppe expandiert werden, weil sie am Ende der Gruppe zurückgesetzt werden. Dafür bietet sich ein `\edef` an. Jedoch müsst ihr mit `\noexpand`/`\unexpanded` und `\expandafter` dafür Sorge tragen, dass nicht zu viel zu früh expandiert wird. Es kann schließlich sein, dass das auszuführende Kommando fragil ist.

`\dospecials` betrifft *nicht* das End-of-Line-Zeichen.

Wenn beim Erstellen der Tokens zwei identische Zeichen von Kategoriecode 7 (Superscript) aufeinander folgen, haben diese eine besondere Bedeutung. Werden sie von zwei *kleingeschriebenen* Hexadezimalen Ziffern gefolgt, werden diese vier Zeichen durch ein Zeichen (*nicht* Token!) mit dem entsprechenden Charactercode ersetzt. Andernfalls, wenn das nächste Zeichen den Charactercode c hat und $c < 128$, dann werden diese drei Zeichen durch ein anderes Zeichen ersetzt, das den Charactercode $c + 64$ hat, wenn $c < 64$, und $c - 64$ andernfalls. So kann das End-of-Line-Character, mit Charactercode 13d, das nach ASCII Code einem Carriage Return entspricht, durch `^^M` erzeugt werden. (Unter der Annahme, dass `\endlinechar` unverändert ist und die üblichen Kategoriecodes gelten.)

Aufgabe 5: Expandable `\alt`

Zahlen, Wenn-Abfragen, Beamer

Schreibt eine vereinfachte, expandierbare Version des `\alt`-Kommandos der `beamer`-Klasse.

`\xalt<`*n*`->{`*then*`}{`*else*`}`

expandiert zu *then* auf allen Overlays $\geq$ *n*, andernfalls *else*. (Der Strich steht für eine von-bis-Angabe, ein offenes Ende steht für den letzten/ersten Overlay.)

Hintergrund

Die `beamer`-Klasse wird zum Erstellen von Bildschirmpräsentationen verwendet. Eine Folie wird in jeweils einer `frame`-Umgebung gesetzt. Mit Hilfe von *Overlays*, die in spitzen Klammern angegeben werden, können Elemente auf einer Folie nach und nach erscheinen.

Um Text-Bausteine nach und nach erscheinen zu lassen, empfiehlt sich das `\uncover`-Kommando, dessen genauer Effekt mit dem `\setbeamercovered`-Kommando eingestellt werden kann. Eine Generalisierung dieses Kommandos ist das `\alt`-Kommando, das sein erstes Argument auf den spezifizierten Folien ausgibt und das zweite Argumente auf den restlichen Folien.

Das original `\alt`-Kommando kann nicht expandierbar sein, da es die Information abspeichern muss, wie viele Overlays der aktuelle Frame hat. Wenn man dieses Feature jedoch weglässt und die Overlays stattdessen separat angibt mit `\begin{frame}<-`*m*`>...` oder mit `\only<-`*m*`>{}`, wird eine expandierbare Version möglich.

Expandierbarkeit ist beispielsweise in Tabellen kritisch. Nicht vollständig expandierbare Tokens führen vor rule/line-Kommandos zu einem `Misplaced \noalign` Fehler und dahinter zu einer potentiell ungewollten weiteren Zeile.

`\xalt` soll es ermöglichen beispielsweise `\bottomrule` und `\multicolumn` overlayaware einzusetzen.

Hinweis

Im original `\alt`-Kommando sind die Overlay Specifications, wie üblich, optional. (Keine Overlay Specifications anzugeben ist äquivalent zu `<*>`, also immer *then*.) Da das `\xalt`-Kommando jedoch nur mit Overlay Specifications Sinn macht, können diese hier als nicht-optional implementiert werden.

Der aktuelle Overlay ist in dem `\beamer@slideinframe` Counter angegeben.

Bonus 1

Erweitere die unterstützten Overlay Specifications um `<`*n*`-`*m*`>`, `<-`*m*`>`, `<`*n*`>` und `<*>`.

Bonus 2

Verwendet `\xalt` um die Kommandos `\toprule`, `\midrule` und `\bottomrule` (aus dem `booktabs`-Paket) umzudefinieren, sodass optional in spitzen Klammern die Overlays angegeben werden können, auf denen die Linien erscheinen sollen. Auf den Overlays, auf denen die Linie nicht sichtbar sind, soll trotzdem Platz für die Linie reserviert werden, damit der Tabelleninhalt nicht springt.

Hinweis

In der `booktabs` Dokumentation in Abschnitt 7 *Technical summary of commands* ist der Platz angegeben, den ein rule-Kommando in Anspruch nimmt.

Hilfe

Beginnt mit einem Spezialfall, wo ihr nur eine Verzweigung benötigt (z. B. Bereich mit offenem Ende) und versucht erstmal nur diesen Spezialfall zu realisieren.

Vorsicht mit `\ifnum`. Nach `\ifnum` (so wie an jeder Stelle, wo TEX eine Zahl erwartet) werden Kommandos expandiert bis die Zahl zu Ende ist. Wenn auf diese Art der then-Block voreilig expandiert wird, kann dies zu unerwünschten Effekten führen.[1] Ihr solltet daher sicher stellen, dass die zweite Zahl des `\ifnum` durch ein Leerzeichen terminiert wird, das beim Lesen der Zahl entsorgt wird. Ein `\relax` funktioniert normalerweise auch anstatt des Leerzeichens, wird aber nicht entsorgt und führt daher in diesem Anwendungsfall eines rule-Kommandos in einer Tabelle zu einem `Misplaced \noalign` Fehler.

Wenn es prinzipiell mit einem Spezialfall funktioniert, erweitert das Kommando Schritt für Schritt um weitere Fälle. Zerlegt das Problem in mehrere Wenn-Abfragen. Überlegt euch anschließend welches TEX-Primitiv geeignet ist um jede der Verzweigung zu realisieren.

Es macht die Sache einfacher, nicht alles in ein Kommando zu quetschen, sondern es auf mehrere Kommandos aufzuteilen (z. B. eines zum Testen ob es sich um eine einzelne Zahl handelt oder um einen Bereich, eines zum Testen ob eine einzelne Zahl dem aktuellen Overlay entspricht, eines zum Testen ob die Untergrenze eines Bereiches beim aktuellen Overlay eingehalten wird und so weiter).

Aus Effizienzgründen sollten diese Kommandos nur die zu testende Overlayspecification als Parameter haben, nicht den *`then`*- und *`else`*-Block. Erst wenn endgültig entschieden ist, ob die Overlayspecification zum aktuellen Overlay passt oder nicht sollten *`then`*- und *`else`*-Block mit Hilfe von `\@firstoftwo` oder `\@secondoftwo` (oder ähnlichen Kommandos) ausgewählt werden. Wenn ein Zwischen-Kommando diese Frage noch nicht abschließend geklärt hat, sollte es *`then`*- und *`else`*-Block unangetastet lassen und stattdessen das nächste (Zwischen-)Kommando aufrufen, das die nächste Entscheidung trifft.

[1] `https://tex.stackexchange.com/questions/495422/ifnum-expanding-too-much-what-is-happening#495425`

Aufgabe 6: `\printcatcodes`

Iterieren über Tokens, Catcodes

Schreibt ein Kommando, das ähnlich `\meaning`, den Replacement Text eines Kommandos ausgibt, aber zusätzlich zu den Zeichen auch die Catcodes anzeigt. Präfixe (`\long`, `\outer`, `\protected`) und Parameterspezifikation brauchen nicht ausgegeben werden.

Hintergrund

Die TEX-Primitive `\meaning`, `\show` und `\tracingmacros` zeigen den Replacementtext von Kommandos und sind sehr nützlich zum Debuggen. Jedoch zeigen sie nicht die Catcodes, was zum Debuggen von Catcode-Spielereien manchmal wünschenswert wäre.

Hinweis

`\the\catcode`\`*`\n`* gibt den Kategoriecode, der aktuell dem Zeichen *n* zugeordnet ist, nicht jedoch den Kategoriecode des Tokens. Solange man keine Catcodes ändert, macht das keinen Unterschied – aber dann bräuchte man auch nicht dieses Kommando. Mit `\ifcat` lässt sich stattdessen vergleichen, ob die Kategoriecodes der beiden folgenden Tokens gleich sind.

Die Zeichen sollten in einer Typewriter Schrift gesetzt werden, damit auch ähnliche Zeichen unterscheidbar sind. Die Zahlen der Catcodes sind in der normalen Schrift jedoch schöner.

Gebt für alle Tokens der Kategorie 1 `\{` aus, für alle Tokens der Kategorie 2 `\}` und für alle Tokens der Kategorie 10 `\textvisiblespace`.

Es wäre zwar theoretisch auch für die Kategorien 1, 2 und 10 möglich das Token über das Expandieren von `\string` in ausgebbare Tokens umzuwandeln, aber wenn es sich um ein Implizites Token handelt (wie z. B. `\bgroup`), dann ist es (soweit ich weiß) unmöglich vorherzusagen, wie viele Tokens `\string` erzeugen würde.

Es ist möglich mit `\if` zu überprüfen, ob das Token denselben Character Code hat wie ein anderes, jedoch hilft das nicht bei Impliziten Tokens. (Wenn `\if \nexttoken {` wahr ist, heißt das nicht, dass das nächste Token `{` ist. Genauso gut kann es auch das Token `\bgroup` sein, das als `\let\bgroup={` definiert ist, s. *The TEXbook* von Donald E. Knuth, Seite 351.)

Mit dem Kommndo `\afterassignment` könnt ihr genau ein Token direkt nach der Ausführung der nächsten Zuweisung (`\def`, `\let`, ...) einfügen.

Hilfe

Beginnt damit, ein Kommando zu schreiben, das alle Tokens bis zu einem Delimiter-Token ohne Expansion ausgibt. Später kann mit `\expandafter` ein-

fach ein Wrapper drumherum geschrieben werden.

Iteriert über jedes Token einzeln, anstatt mehrere mit einem Delimited Parameter einzulesen.

Verwendet `\futurelet` anstatt eines Parameters um das Auffressen von Space- und Grouptokens zu vermeiden und potentiellen Problemen mit `\if`-Statements vorzubeugen.

Denkt daran, dass `\futurelet`, im Gegensatz zu `\let`, das kopierte Token *nicht* aus der Liste der zu verarbeitenden Tokens löscht.

Abhängig von der Kategorie können nicht alle Tokens gleich behandelt werden.

Bedenkt, dass `\ifcat` Tokens expandiert. Wenn ihr das nicht möchtet, könnt ihr `\noexpand` verwenden.

Es empfiehlt sich möglichst viel mit expliziten Tokens (also `#1` statt `\nexttoken`) zu arbeiten, weil deren Repräsentation im Source Code mit Hilfe von `\string` wiederhergestellt werden kann.

Tokens der Kategorien 1 (Begin Group), 2 (End Group) und 10 (Space) können aber nicht direkt als Argument gelesen werden und bedürfen daher einer Extrabehandlung.

Bei der Verwendung von `\let` dürfen optional zwischen dem zu definierenden Token und dem zu kopierenden Token ein Gleichheitszeichen und hinter dem Gleichheitszeichen ein Space Token stehen. Wenn man mit `\let` ein Space Token zuweisen möchte, so muss sich zwischen diesem Space Token und dem Gleichheitszeichen genau ein weiteres Space Token befinden, damit das zuzuweisende Space Token nicht ignoriert wird.

Aufgabe 7: `csvtable`

Ändern von Catcodes, Expandierbarkeit

Erstellt eine Umgebung `csvtable`, in der Comma Separated Values eingefügt werden können.

Beispiel

```
\begin{csvtable}{ll}%
        \toprule%
        \input{data.csv}%
        \bottomrule%
\end{csvtable}%
```

Bonus 1

Auch ohne das Auskommentieren der Zeilenenden im obigen Beispiel sollen keine ungewollten Leerzeilen eingefügt werden. Ebenso soll auch nach `\midrule` ein aktives EOL Zeichen ignoriert werden.

Bonus 2

Definiert ein Kommando `\setcsvtableenvironment`, mit dem geändert werden kann, welche Umgebung zum Setzen der Tabelle verwendet werden soll.

Als erforderliches Argument soll das Kommando den Namen der Umgebung erwarten.

Mit einem weiteren, optionalen Argument, soll der Umgebung ein möglicherweise erforderliches Argument mitgegeben werden können:

```
\setcsvtableenvironment{tabular*}[.4\linewidth]
```

Bonus 3

Leerzeilen, insbesondere am Ende der Datei, sollen ignoriert werden.

Hinweis

`^^M` ist das End-of-Line Zeichen.

Das Verarbeiten von u. a. CSV-Dateien geht mit dem `datatool` Paket flexibler, ist aber rechenaufwändiger.

Zwischen `\\` und rule/line Kommandos dürfen keine nicht expandierbaren Tokens vorkommen. Ebenso dürfen auch nach `\bottomrule` keine nicht expandierbaren Token stehen, ansonsten wird dort eine weitere Zeile eingefügt.

`\input` hat aus Sicht von TEX kein Argument, sondern überprüft nur, ob das das nächste Zeichen eine geschweifte Klammer ist. Falls nicht, wird direkt das TEX-Primitiv `\input`, das in LATEX `\@@input` heißt, verwendet. Falls ja, wird der LATEX-Wrapper `\@iinput` verwendet, der erstmal überprüft, ob die Datei existiert, bevor `\@@input` verwendet wird.

Hilfe

Ändert die Catcodes von `,` und `^^M`.

Die Änderungen der Catcodes müssen vor der Tabelle passieren, weil jede Zelle eine eigene Gruppe ist und die Änderungen damit am Ende der Zelle zurückgesetzt werden würden.

Ändert den Catcode von `,` auf Alignment Tab und den Catcode von `^^M` auf Active.