The grading-scheme package

Maximilian Keßler*

Released 2022/02/24

version 0.1.1

Abstract

This package aims at an easy-to-use interface to typeset grading schemes in tabular format, in particular grading-schemes of exercises of mathematical olympiads where multiple solutions have to be graded and might offer mutual exclusive ways of receiving points.

Contents

---

*ctan@maximilian-kessler.de

SUM				At least one correct solution	**1 P.**
				All correct solutions	**1 P.**
	MAX	Uniqueness proofs			
		MAX	First solution		
			SUM	Show that either $4 \mid p-2$ or $3 \mid p-4$.	**3 P.**
				Show that $m = 5$	**2 P.**
				Show that $n = 4$	**1 P.**
				Conclude solutions	**1 P.**
			Second solution		
			SUM	Show that $p$ is even	**2 P.**
				Reduce to at most 30 candidates	**3 P.**
				Test all candidates	**2 P.**
		MIN(2)	Plausible Arguments		
			SUM	plausible argument that there are finitely many solutions	**1 P.**
				suggestion that divisibility by 4 is useful	**1 P.**
				suggesting that not both of m, n can be 'large'.	**1 P.**

Table 1: Example output of a grading scheme.

1 General notions

Probably, an example is fitting best at this point. If you want to set up a grading scheme, with the `grading-scheme` package all you have to do to get the output of Table 1 is type the following:

```
\begin{gradingscheme}{sum}
  | At least one correct solution & 1
  | All correct solutions & 1
  \begin{block}{max}
    \begin{block}[Uniqueness proofs]{max}
      \begin{block}[First solution]{sum}
        | Show that either $4 \mid p - 2$ or  $3 \mid p - 4$. & 3
        | Show that  $m = 5$ & 2
        | Show that $ n = 4 $ & 1
        | Conclude solutions & 1
      \end{block}
      \begin{block}[Second solution]{sum}
        | Show that $p$ is even & 2
        | Reduce to at most 30 candidates & 3
        | Test all candidates & 2
      \end{block}
    \end{block}
    \begin{block}{min(2)}
      \begin{block}[Plausible Arguments]{sum}
        | plausible argument that there are finitely many solutions & 1
        | suggestion that divisibility by 4 is useful & 1
        | suggesting that not both of $m$,  $n$ can be 'large'. & 1
      \end{block}
    \end{block}
  \end{block}
\end{gradingscheme}
```

Note in particular that the correct width of the rows and columns are automatically adjusted.

For us, a grading scheme consists of several statements, each worth some specific number of points, that will be combined into a final number of points using a nested expression of operations, typically sums, maximums or minimus.

An abstract grammar defining our notions is defined in Table 2. The meanings of ⟨*balanced text*⟩ and ⟨*number*⟩ are as usual.

grading scheme := block
block := [⟨*block description*⟩], ⟨*block operation*⟩, ⟨*element list*⟩
element list := ⟨*element*⟩ | ⟨*element*⟩ : ⟨*element list*⟩
element := ⟨*block*⟩ | ⟨*entry*⟩
entry := ⟨*entry description*⟩, ⟨*entry points*⟩
block description := ⟨*balanced text*⟩
block operation := ⟨*balanced text*⟩
entry description := ⟨*balanced text*⟩
entry points := ⟨*number*⟩

Table 2: Abstract grammer for a grading scheme

The package facilitates specifying a grading scheme according to this grammar and typesets these.

2 Usage

Load the package as usual by

```
\usepackage[pipe]{grading-scheme}
```

The `pipe` option is of course optional and the only currently supported option. It activates the pipe syntax explained later.

`gradingscheme`

```
\begin{gradingscheme}[⟨description text⟩]{⟨operator⟩}
\end{gradingscheme}
```

The main environment provided by this package. Inside the grading scheme, any number of blocks and entries can be specified. The grading scheme environment will act like the outer-most block and typeset all its contents.

Grading schemes cannot contain each other. As they are implemented as `tabular`s, you can (and maybe should) put a grading scheme into a `table` environment as if it were a regular `tabular`.

TEXhackers note: Actually, the gradingscheme environment just sets up some hooks that further block environments and entries will fill and typesets these at the end of the environment.

You could technically put any regular text inside the grading scheme and this will be typeset as if it had been specified before the grading scheme. However, this is definitely not recommended.

`block`

`\begin{block}[⟨description text⟩]{⟨operator⟩}`
`\end{block}`

The ⟨*description text*⟩ will be shown in a separate row on top of the block. If not specfied, no such row is inserted. The ⟨*operator*⟩ is printed in small capitals (and rotated) in a column that spans the left side of the block, indicating that this operation is applied to the block.

The block will be an element of the outer block or grading scheme it is contained in.

Inside the block, any number of blocks or entries can be specified (in order) that will be typeset as members of this block.

A block environment can only be used inside a gradingscheme. Block environments can contain each other in balanced form.

`\entry` `\entry{⟨entry description⟩}{⟨points⟩}`

Specifies an entry with given description and points. The entry is typeset as part of the block/grading scheme containing it.

`|` `| ⟨entry description⟩ & ⟨points⟩ \n`

This is an alternative form to specify an entry. The `\n` is to be interpreted as a newline in the source file, so | reads until the end of the line.

This is equivalent to calling `\entry{`⟨*entry description*⟩`}{`⟨*points*⟩`}`

The package option `pipe` has to be specified to activate this syntax.

TEXhackers note: The | character (pipe) is made an active character for the scope of each gradingscheme environment. Thus, this syntax only works if the gradingscheme is read in expansion mode by LaTeX, since otherwise the category code will be messed up.

3 LaTeX3 interface

There is also a LaTeX3 interface for dealing with this package that can deal with all of the mentioned notions separately, technically allowing you to combine them in other ways or formatting them into your custom tables.

These functions are for now not documented separately, since the author thinks that such use cases would be rather rare.

The LaTeX3 interface is, however, *not* considered to be stable yet, so if you really have to rely on this, feel free to contact the author.

Change History

0.1

General: Initial release 1

0.1.1

General: Bug fix (thanks to Denis Bitouzé), typos. 1

4 grading-scheme implementation

```
⟨*package⟩
⟨@@=grading_scheme⟩
```

4.1 Dependencies and options

Currently, there is only one option

\g__grading_scheme_pipe_syntax_bool This will toggle the pipe syntax option of the package.

```
\bool_new:N \g__grading_scheme_pipe_syntax_bool
```

(*End definition for* `\g__grading_scheme_pipe_syntax_bool`*.*)

```
\keys_define:nn { grading-scheme }
  {
    pipe    .bool_gset:N =  \g__grading_scheme_pipe_syntax_bool,
    pipe    .default:n  = { true },
    unknown .code:n       =
      {
        \msg_error:nnn { grading-scheme } { unknown-option} { \l_keys_key_str }
      }
  }
\msg_new:nnn { grading-scheme } { unknown-option}
  {
    Unknown ~ option ~ '#1'.
  }
\RequirePackage { l3keys2e }
\ProcessKeysOptions { grading-scheme }
\RequirePackage{multirow}
\RequirePackage{rotating}
```

4.2 A simple logging module

```
⟨*log⟩
```

We provide a simple log/debugging facility for this package.

\g__grading_scheme_log_depth_int Stores the log depth for indentation in the log file.

```
\int_new:N \g__grading_scheme_log_depth_int
```

(*End definition for* `\g__grading_scheme_log_depth_int`*.*)

__grading_scheme_log_incr: Increments the log depth

```
\cs_new:Npn \__grading_scheme_log_incr:
  {
    \int_gincr:N \g__grading_scheme_log_depth_int
  }
```

(*End definition for* `\__grading_scheme_log_incr:`.)

`\__grading_scheme_log_decr:`

Decrements the log depth

```
\cs_new:Npn \__grading_scheme_log_decr:
  {
    \int_gdecr:N \g__grading_scheme_log_depth_int
  }
```

(*End definition for* `\__grading_scheme_log_decr:`.)

`\__grading_scheme_log:n`
`\__grading_scheme_log:x`

Logs to the terminal and log file using the current depth.

```
\cs_new:Npn \__grading_scheme_log:n #1
  {
    \iow_term:x
      {
        [gs] \prg_replicate:nn \g__grading_scheme_log_depth_int { . . }
        \exp_not:n { #1 }
      }
    \iow_log:x
      {
        [gs] \prg_replicate:nn \g__grading_scheme_log_depth_int { . . }
        \exp_not:n { #1 }
      }
  }
\cs_generate_variant:Nn \__grading_scheme_log:n { x }
```

(*End definition for* `\__grading_scheme_log:n`.)

```
⟨/log⟩
```

4.3 Wrappers

We provide some LaTeX3 wrappers for private usage.

`\__grading_scheme_multicolumn:nnn`

Just a wrapper around `\multicolumn` from LaTeX 2_ε.

```
\cs_set_eq:NN \__grading_scheme_multicolumn:nnn \multicolumn
```

(*End definition for* `\__grading_scheme_multicolumn:nnn`.)

`\__grading_scheme_multirow:nnn`

Just a wrapper around `\multirow` from LaTeX 2_ε.

```
\cs_set_eq:NN \__grading_scheme_multirow:nnn \multirow
```

(*End definition for* `\__grading_scheme_multirow:nnn`.)

`\__grading_scheme_cline:n`
`\__grading_scheme_cline:nn`

A wrapper around `\cline`.

The second form takes the beginning and ending column as separate arguments.

```
\cs_set_eq:NN \__grading_scheme_cline:n \cline
\cs_new:Npn \__grading_scheme_cline:nn #1 #2
  {
    \__grading_scheme_cline:n { #1 - #2 }
  }
```

(*End definition for* `\__grading_scheme_cline:n` *and* `\__grading_scheme_cline:nn`.)

`\__grading_scheme_hline:` Wrapper for a `\hline`

```
\cs_set_eq:NN \__grading_scheme_hline: \hline
```

(*End definition for* `\__grading_scheme_hline:`.)

`\__grading_scheme_rotatebox:nnn` Wrapper around rotatebox

```
\cs_new:Npn \__grading_scheme_rotatebox:nnn #1 %implicit #2, #3
  {
    \rotatebox [ #1 ]
  }
```

(*End definition for* `\__grading_scheme_rotatebox:nnn`.)

4.4 Customizable backends

We set up some functions that will be used internally, but could technically be customized at some point. For now, these are just constants.

`\__grading_scheme_points:n` Prints the number of points. Used in the last column of the grading scheme.

```
\cs_new:Npn \__grading_scheme_points:n #1
  {
    \textbf{ #1 ~ P. }
  }
```

(*End definition for* `\__grading_scheme_points:n`.)

`\__grading_scheme_operation:n` Prints an operation. Used in the leftmost column of a block.

```
\cs_new:Npn \__grading_scheme_operation:n #1
  {
    { \sc #1 }
  }
```

(*End definition for* `\__grading_scheme_operation:n`.)

4.5 Resources

4.5.1 Some general thoughts

This is some general reasoning about why we model data structures as we do in this package, comparing parts of LaTeX3 to plain `C` code.

We have to model elements, blocks and entries here. LaTeX3 provides some data structures for this, and we will use property-lists to model our data structures for this package.

When thinking about what a property list is, a call to `\prop_new:N` essentially allocates some memory for us and makes our given argument a *pointer* to a data structure that we refer to as a "property list".

We want to think of these as pointers since modifying a property list at some place really also modifies this property list at another, so these behave like pointers.

Considering the grouping mechanism of TeX, this is of course not quite right for local variables (we assume that in the LaTeX3 spirit, all variables are either local or global once and for all, to avoid confusion) since these rather correspond to one pointer at each grouping level, where upon entering such a grouping level, the (new) pointer is initialized with a copy of the outer instance of the pointer and destroyed again when leaving the current grouping level.

In this spirit, we really want to think of grouping levels as *scopes* like in `C`.

Considering functions now, a typical LaTeX3 function has no return value (these would be functions with the `_p` predicate or `_use:` functions that leave contents in the input stream), since this is in general not desired. Rather, we pass pointers as a function argument and receive the desired "return value" as written into our given pointer (overwriting any prior data), which is also a common behaviour in large parts of `C`.

Now, as mentioned earlier, data structures such as property queues are just pointers that refer to some tokens with specific internal structure. All functions dealing with such pointers have to ensure that these invariants remain satisfied, and this is what functions like `\prop_get:NnN` or `\prop_put:Nn` do. The key point here is that we refer to these structures only with pointers and do not deal with them directly. This is what gives us some very nice abstrict model, hiding the class invariants of a property queue from the user and providing high-level semantic access to such data structures.

Just for completeness, we mention that this is not the case for *all* data structures. For example, a `clist` has some structure that is well-known and can be dealt with by the user directly. That is why there are also `clist` commands acceptiong `n`-type arguments for dealing with a `clist`, but no such commands for property-lists, as they only use `N` and thus essentially pointers.

If we want to model data structures now, especially ours from this package, we even want data structures containing other data structures. Although technically, storing all data of a grading scheme in a single macro would be possible, due to its recursive structure, parsing would be quite a challenge and possibly also rather slow. Also, this would require *direct* access to the representation of blocks contained in this block, which is undesired.

Compare this to the question of putting a property-list inside a property-list. Since there is no (semantic/provided) possibility of actually putting a property-list anywhere, we cannot store property-lists in property-lists directly. We can, however, store the name of a property list inside of another one, this amounts to actually storing the *pointer* to the first list inside the second one.

Now, back to our package, we essentially want to model blocks and alike as `C`-style `struct`s that contain *pointers* to their elements and similar. The classic question of ownership arises in this context, so we would have to consider shallow or deep copies and a user has to watch out when dealing with our data structures.

Since only a limited set of operations is needed in our case, we take a simple approach by saying that each struct owns all its contents, that is, owns everythin its data member pointers point to, also recursively. This essentially forbids shallow-copying (violating our assumption) but we do not need this here, so this is okay.

Still, this hase some undesired consequences: Since in LaTeX3 each pointer has some name, we need to construct a lot of pointers on the fly where we are not able to actually specify their names directly, so we have to indirectly deal with pointers. That means that we actually need double-pointers, i.e. local variables *containing* pointers to the data structures we are actually interested in. This is what a `void**` would correspond to in `C`. Also, note that once we store a pointer to some element in a property queue (e.g. the name of the list of members when dealing with a block), we actually also need a `void**` when retrieving this value again from the property queue, since we return by pointer.

Consequently, this leads to quite a lot of expansion-control when dealing with such pointers, and if not taking precautions, also to easy confusion about what is a pointer and what not.

Our approach to tackle these problems is described in subsection 4.6.

4.6 Pointers

This module should/will be reworked into an own package in the future. Also, there is no proper documentation for it apart from the comments in the implementation, since this is not meant to be used in other code than this package currently, nor regarded stable.

We introduce a new data type of `pointer` and a new argument type of `P`. The argument type `P` shall indicate a single expansion *with the assumption that this expansion will yield a single token*. So, from an implementation point of view, this is the same as an `o`-type argument, just with further restricted assumptions.

This also enables us to generate `P` variant arguments from `N` variant macros. Compare this to the usual deprecation warning that LaTeX3 gives when generating an `o` variant from an `N`-type argument:

```
! LaTeX3 Error: Variant form 'o' deprecated for base form '\foo:N'. One
(LaTeX3)        should not change an argument from type 'N' to type 'o':
(LaTeX3)        base form only accepts a single token argument.
```

Since basically anything in LaTeX3 is in fact a pointer, we will introduce the `pointer` structure with the `ptr` type (suffix) and actually mean that this is a `void**`.

Thus, we introduce the following conventions:

1. A TeX-pointer is a TeXcontrol sequence that represents some data.
2. A `pointer` is a control sequence that expands to a single TeX control sequence that is a TeX-pointer.
3. The *name* of a `pointer` is the name of the control sequence representing it. Thus, such a name always ends with `_ptr` by convetion.
4. A `ptr` is said to be `NULL` if its expansion yields an undefined TeX control sequence. We also refer to this as a `nullptr`.
5. A *typed pointer* is a pointer where the underlying type is specified. By this, we mean that expanding the token exactly once yields a token representing a data structure of the type we said this pointer to have or that the pointer is `NULL`.
6. A *void pointer* shall be a pointer whose actual type is unknown.
7. When a `ptr` is of type `type`, we denote these by the suffix `_type_ptr`.
8. *Dereferencing* a pointer amounts to expanding it exactly once.
9. A `P`-type argument accepts a single token that has to be of type `ptr`. The token is expanded exactly once and then treated as an `N`-type argument in the corresponding function.

`\g__ptr_counter_int` Counts the number of pointers given by `\ptr_new:N`. This ensures that each pointer can have a unique name regardless of grouping.

```
\int_new:N \g__ptr_counter_int
```

(*End definition for* `\g__ptr_counter_int`.)

`\l__ptr_var_prefix_str` Stores the prefix of new pointer variables.

```
\str_new:N \l__ptr_var_prefix_str
```

(*End definition for `\l__ptr_var_prefix_str`.*)

`\ptr_new:N` `\ptr_new:N`⟨*pointer*⟩

Gives a new unique pointer that is `NULL`. Gives an error when the ⟨*pointer*⟩ already exists. The pointer variable is created globally.

```
\cs_new:Npn \ptr_new:N #1
  {
    \__ptr_get_var_prefix:NN #1 \l__ptr_var_prefix_str
    \ptr_new:NVn #1 \l__ptr_var_prefix_str { any }
  }
\cs_new:Npn \ptr_new:Nn #1 #2
  {
    \ptr_new:Nnn { #1 } { #2 } { any }
  }
% ptr var, prefix, type
\cs_new:Npn \ptr_new:Nnn #1 #2 %i #3
  {
    \str_case:nnT
      { #2 }
      {
        { l } { }
        { g } { }
      }
      {
        \tl_new:N #1
      }
    \__ptr_init:Nnn #1 { #2 } %i { #3 }
  }
\cs_generate_variant:Nn \ptr_new:Nnn { N V n }
% ptr var, prefix, type
% assumes that the pointer name exists already in case of l or g prefix
\cs_new:Npn \__ptr_init:Nnn #1 #2 % implicit #3
  {
    \__ptr_get_var_prefix:NN #1 \l__ptr_var_prefix_str
    \str_case:VnF
      \l__ptr_var_prefix_str
      {
        { l }
        {
          \cs_set_eq:NN \__ptr_poly_tl_set:Nx \tl_set:Nx
        }
        { g }
        {
          \cs_set_eq:NN \__ptr_poly_tl_set:Nx \tl_gset:Nx
        }
        { c }
        {
          \cs_set_eq:NN \__ptr_poly_tl_set:Nx \tl_const:Nx
        }
      }
      {
        \str_show:N \l__ptr_var_prefix_str % TODO: show error
      }
    \__ptr_init_aux:Nnn #1 { #2 }
  }
\cs_generate_variant:Nn \__ptr_init:Nnn { N V n }
% ptr var, prefix, type. assumes poly_tl_set:Nx has been set
\cs_new:Npn \__ptr_init_aux:Nnn #1 #2 #3
  {
    \__ptr_poly_tl_set:Nx #1
      {
        \exp_not:c
          {
            #2
            __ptr__unique__
            \int_use:N \g__ptr_counter_int
            _
            #3
          }
      }
    \int_gincr:N \g__ptr_counter_int
  }
\cs_generate_variant:Nn \__ptr_init:Nnn { N V n }
```

(*End definition for* `\ptr_new:N`*. This function is documented on page* **??**.)

`\__ptr_get_var_prefix:NN`

Gets the first character of the name of the variable given as `#1`. The first character is assumed to be one of `c`, `l` or `g` by usual naming conventions.

```
\cs_new:Npn \__ptr_get_var_prefix:NN #1 #2
  {
    \tl_if_head_eq_charcode:fNTF { \cs_to_str:N #1 } l
      {
        \str_set:Nn #2 { l }
      }
      {
        \tl_if_head_eq_charcode:fNTF { \cs_to_str:N #1 } g
          {
            \str_set:Nn #2 { g }
          }
          {
            \tl_if_head_eq_charcode:fNTF { \cs_to_str:N #1 } c
              {
                \str_set:Nn #2 { c }
              }
              {
                \msg_error:nnx { ptr } { variable-name-illegal-prefix }
                  {
                    \token_to_str:N #1
                  }
              }
          }
      }
  }
  \msg_new:nnnn { ptr } { variable-name-illegal-prefix }
    {
      Requested ~ new ~ pointer ~ '#1' ~ has ~ illegal ~ prefix.
    }
    {
      Prefix ~ should ~ be ~ one ~ of ~ 'l', ~ 'g' ~ or ~ 'c'.
    }
```

(*End definition for* `\__ptr_get_var_prefix:NN`*.*)

`\ptr_clear:N` Clears this pointer. This makes the pointer equivalent to a new `nullptr`.

```
\cs_new:Npn \ptr_clear:N #1
  {
    \__ptr_get_var_prefix:NN #1 \l__ptr_var_prefix_str
    \__ptr_init:NVn #1 \l__ptr_var_prefix_str { any }
  }
\cs_new:Npn \ptr_clear:Nn #1 #2
  {
    \__ptr_init:Nnn #1 { #2 } { any }
  }
```

(*End definition for* `\ptr_clear:N`*. This function is documented on page* **??***.*)

`\ptr_set:NN` Sets the pointer to represent the given argument.

```
\cs_set_eq:NN \ptr_set:NN \tl_set:Nn
```

(*End definition for* `\ptr_set:NN`*. This function is documented on page* **??***.*)

`\ptr_set_eq:NN` Accepts two pointers. The first is set to be equivalent to the second.

```
\cs_set_eq:NN \ptr_set_eq:NN \tl_set_eq:NN
```

(*End definition for* `\ptr_set_eq:NN`*. This function is documented on page* **??***.*)

`\ptr_use:N` Uses the pointer, i.e. expands to the stored TEX-pointer.

```
\cs_set_eq:NN \ptr_use:N \tl_use:N
```

(*End definition for* `\ptr_use:N`*. This function is documented on page* **??***.*)

`\__ptr_check:N` Checks that the given argument is a pointer.

```
\cs_new:Npn \__ptr_check:N #1
  {
    \tl_if_single:NF #1
      {
        \msg_error:nnx { ptr } { is-not-a-pointer }
          {
            \token_to_str:N #1
          }
      }
  }
\msg_new:nnn { ptr } { is-not-a-pointer }
  {
    The ~ control ~ sequence ~ '#1' ~ is ~ not ~ a ~ valid ~ pointer.
  }
```

(*End definition for* `\__ptr_check:N`.)

`\__ptr_check:NN`

Checks that the second argument is a pointer.

```
\cs_new:Npn \__ptr_check:NN #1 #2
  {
    \__ptr_check:N #2
    #1 #2
  }
```

(*End definition for* `\__ptr_check:NN`.)

`\__ptr_check:nN`

Checks that the second argument is a pointer.

```
\cs_new:Npn \__ptr_check:nN #1 #2
  {
    \__ptr_check:N #2
    { #1 } #2
  }
```

(*End definition for* `\__ptr_check:nN`.)

`\l__ptr_content_tl`

Stores the (internal) TeX-pointer of a `pointer`.

```
\tl_new:N \l__ptr_content_tl
```

(*End definition for* `\l__ptr_content_tl`.)

`\exp_args:NP`
`\exp_args:NNP`
`\exp_args:NNNP`
`\exp_args:NNNNP`
`\exp_args:NPP`

Expands a pointer. This leaves the first tokens as a single token in the input stream, followed by the value of the pointer as a single token.

If the last argument does not expand to a single token, an error is given.

```
\cs_new:Npn \exp_args:NP #1 #2
  {
    \__ptr_check:N #2
    \exp_last_unbraced:No #1 #2
  }
\cs_new:Npn \exp_args:NNP #1 #2 #3
  {
    \__ptr_check:N #3
    \exp_last_unbraced:NNo #1 #2 #3
  }
\cs_new:Npn \exp_args:NNNP #1 #2 #3 #4
  {
    \__ptr_check:N #4
    \exp_last_unbraced:NNNo #1 #2 #3 #4
  }
\cs_new:Npn \exp_args:NNNNP #1 #2 #3 #4 #5
  {
    \__ptr_check:N #5
    \exp_last_unbraced:NNNNo #1 #2 #3 #4 #5
  }
\cs_new:Npn \exp_args:NPP #1 #2 #3
  {
    \__ptr_check:N #2
    \__ptr_check:N #3
    \exp_last_two_unbraced:Noo #1 #2 #3
  }
```

(*End definition for* `\exp_args:NP` *and others. These functions are documented on page* **??**.)

`\ptr_if_null:NTF`

Checkes if the pointer is `NULL`.

```
\cs_new:Npn \ptr_if_null:NT
  {
    \exp_args:NP \cs_if_exist:NF
  }
\cs_new:Npn \ptr_if_null:NF
  {
    \exp_args:NP \cs_if_exist:NT
  }
\cs_new:Npn \ptr_if_null:NTF #1 #2 #3
  {
    \exp_args:NP \cs_if_exist:NTF #1 { #3 } { #2 }
  }
```

(*End definition for* `\ptr_if_null:NTF`*. This function is documented on page* **??**.)

`\tl_set_eq:NP`
`\tl_set_eq:PN`
`\tl_set_eq:PP`
`\tl_use:P`
`\exp_not:P`
`\clist_new:P`
`\clist_put_right:Pn`
`\clist_gput_right:Pn`
`\clist_map_function:PN`
`\prop_show:P`
`\clist_show:P`

Just variants of `\tl_set_eq:NN` with indicated signature.

```
\cs_new:Npn \tl_set_eq:NP
  {
    \exp_args:NNP \tl_set_eq:NN
  }
\cs_new:Npn \tl_set_eq:PN
  {
    \exp_args:NP \tl_set_eq:NN
  }
\cs_new:Npn \tl_set_eq:PP
  {
    \exp_args:NPP \tl_set_eq:NN
  }
\cs_new:Npn \tl_use:P
  {
    \exp_args:NP \tl_use:N
  }
\cs_new:Npn \exp_not:P
  {
    \exp_args:NP\exp_not:N
  }
\cs_new:Npn \clist_new:P
  {
    \exp_args:NP \clist_new:N
  }
\cs_new:Npn \clist_show:P
  {
    \exp_args:NP \clist_show:N
  }
\cs_new:Npn \clist_put_right:Pn
  {
    \exp_args:NP \clist_put_right:Nn
  }
\cs_new:Npn \clist_gput_right:Pn
  {
    \exp_args:NP \clist_gput_right:Nn
  }
\cs_new:Npn \clist_map_function:PN
  {
    \exp_args:NP \clist_map_function:NN
  }
\cs_new:Npn \prop_show:P
  {
    \exp_args:NP \prop_show:N
  }
```

(*End definition for* `\tl_set_eq:NP` *and others. These functions are documented on page* **??**.)

`\ptr_show:N` Shows information about this pointer, namely:

If it is `NULL`, then it indicates this. If it is not `NULL`, then the TEX-pointer and the contained contents of the TEX-pointer (in unexpanded form) are shown.

```
\cs_new:Npn \ptr_show:N #1
  {
    \__ptr_check:N #1
    \ptr_if_null:NTF #1
      {
        \tl_show:x
          {
            \token_to_str:N #1 -> \exp_not:P #1 = NULL
          }
      }
      {
        \tl_show:x
          {
            \token_to_str:N #1 -> \exp_not:P #1 -> \exp_args:NP \exp_not:V #1
          }
      }
  }
```

(*End definition for* `\ptr_show:N`*. This function is documented on page* **??**.)

4.7 Structs

We will model `C`-style `struct`s as property-lists in this package. Each struct member is put into the property list, using its 'name' as a key, and we store the corresponding contents there.

We will also have a local variable named correspondingly for each `struct` member. In case we deal with a struct and want to acces its data, we load the values from the property-list into these local variables. Thus, the full information about a `struct` is contained in the property-list, and we can work with them quite conveniently when implementing functions.

4.8 Entries

An entry is for us the following:

```
struct Entry {
  tl description;
  int points;
}
```

\grading_scheme_entry_new:N
\grading_scheme_entry_clear:N
\grading_scheme_entry_set_eq:NN
\grading_scheme_entry_gclear:N
\grading_scheme_entry_gset_eq:NN

Do what their signature suggests. We just forward them to the property lists.

```
\cs_set_eq:NN \grading_scheme_entry_new:N      \prop_new:N
\cs_set_eq:NN \grading_scheme_entry_clear:N    \prop_clear:N
\cs_set_eq:NN \grading_scheme_entry_set_eq:NN  \prop_set_eq:NN
\cs_set_eq:NN \grading_scheme_entry_gclear:N   \prop_gclear:N
\cs_set_eq:NN \grading_scheme_entry_gset_eq:NN \prop_gset_eq:NN
```

(*End definition for* `\grading_scheme_entry_new:N` *and others. These functions are documented on page* **??**.)

\l__grading_scheme_entry_points_int
\l__grading_scheme_entry_description_tl

The mentioned local variables where we retrieve values.

```
\int_new:N \l__grading_scheme_entry_points_int
\tl_new:N  \l__grading_scheme_entry_description_tl
```

(*End definition for* `\l__grading_scheme_entry_points_int` *and* `\l__grading_scheme_entry_description_tl`.)

\grading_scheme_entry_set_description:Nn
\grading_scheme_entry_gset_description:Nn

Sets the description.

```
\cs_new:Npn \grading_scheme_entry_set_description:Nn #1 % implicit description
  {
    \prop_put:Nnn #1 { description }
  }
\cs_new:Npn \grading_scheme_entry_gset_description:Nn #1 % implicit description
  {
    \prop_gput:Nnn #1 { description }
  }
```

(*End definition for* `\grading_scheme_entry_set_description:Nn` *and* `\grading_scheme_entry_gset_description:Nn`. *These functions are documented on page* **??**.)

\grading_scheme_entry_set_points:Nn
\grading_scheme_entry_gset_points:Nn

Sets the points.

```
\cs_new:Npn \grading_scheme_entry_set_points:Nn #1 #2
  {
    \prop_put:Nnx #1 { points } { \int_eval:n { #2 } }
  }
\cs_new:Npn \grading_scheme_entry_gset_points:Nn #1 #2
  {
    \prop_gput:Nnn #1 { points } { \int_eval:n { #2 } }
  }
```

(*End definition for* `\grading_scheme_entry_set_points:Nn` *and* `\grading_scheme_entry_gset_points:Nn`. *These functions are documented on page* **??**.)

\grading_scheme_entry_get_description:NN

Gets the description of the entry and stores it in `#2`. The assignment is local.

```
\cs_new:Npn \grading_scheme_entry_get_description:NN #1 %implicit #2
  {
    \prop_get:NnN #1 { description }
  }
```

(*End definition for* `\grading_scheme_entry_get_description:NN`. *This function is documented on page* **??**.)

\grading_scheme_entry_get_points:NN

Gets the points of the entry and stores it in #2. The assignment is local.

```
\cs_new:Npn \grading_scheme_entry_get_points:NN #1 %implicit #2
  {
    \prop_get:NnN #1 { points }
  }
```

(*End definition for* `\grading_scheme_entry_get_points:NN`. *This function is documented on page* **??**.)

__grading_scheme_entry_load_points:N

Loads the points into the local variable

```
\cs_new:Npn \__grading_scheme_entry_load_points:N #1
  {
    \grading_scheme_entry_get_points:NN #1 \l__grading_scheme_entry_points_int
  }
```

(*End definition for* `\__grading_scheme_entry_load_points:N`.)

__grading_scheme_entry_load_description:N

Loads the description of an entry

```
\cs_new:Npn \__grading_scheme_entry_load_description:N #1
  {
    \grading_scheme_entry_get_description:NN #1 \l__grading_scheme_entry_description_tl
  }
```

(*End definition for* `\__grading_scheme_entry_load_description:N`.)

\grading_scheme_entry_format:NnnN
\grading_scheme_entry_gformat:NnnN
\grading_scheme_entry_format:PnnN
\grading_scheme_entry_gformat:PnnN

`\grading_scheme_entry_put:NnnN{⟨entry⟩}{⟨indent⟩}{⟨width⟩}{⟨tl var⟩}`

Puts the formatted contents of the entry into the ⟨*tl var*⟩. The ⟨*indent*⟩ specify how many tabs will be inserted before adding the contents. The ⟨*width*⟩ specifies how many columns this entry will occupy. This ⟨*width*⟩ has to be at least 2.

An \\ is inserted at the end of the entry.

```
\cs_new:Npn \grading_scheme_entry_format:NnnN
  {
    \cs_set_eq:NN \__grading_scheme_tl_put_right:Nx \tl_put_right:Nx
    \__grading_scheme_entry_format:NnnN
  }
\cs_new:Npn \grading_scheme_entry_gformat:NnnN
  {
    \cs_set_eq:NN \__grading_scheme_tl_put_right:Nx \tl_gput_right:Nx
    \__grading_scheme_entry_format:NnnN
  }
\cs_new:Npn \grading_scheme_entry_format:PnnN
  {
    \exp_args:NP \grading_scheme_entry_format:NnnN
  }
\cs_new:Npn \grading_scheme_entry_gformat:PnnN
  {
    \exp_args:NP \grading_scheme_entry_gformat:NnnN
  }
```

(*End definition for* `\grading_scheme_entry_format:NnnN` *and others. These functions are documented on page* **??**.)

__grading_scheme_entry_format:NnnN
__grading_scheme_entry_format:PnnN

Aux function that assumes that `\__grading_scheme_tl_put_right:Nx` has been so to globally or locally putting into the token list.

```
\cs_new:Npn \__grading_scheme_entry_format:NnnN #1 #2 #3 #4
  {
⟨*log⟩
    \__grading_scheme_log:x
      { Formatting ~ entry ~ '\token_to_str:N #1' ~ into ~ '\token_to_str:N #4' with }
    \__grading_scheme_log:n { indent = '#2' ~ and ~ width ~ '#3' }
    \prop_log:N #1
    \__grading_scheme_log_incr:
⟨/log⟩
    \__grading_scheme_entry_assert_description:N #1 % implicitly loads value
    \__grading_scheme_entry_assert_points:N #1      % implicitly loads value
    \__grading_scheme_tl_put_right:Nx #4
      {
        \prg_replicate:nn { #2 } { & }
        \exp_not:N \__grading_scheme_multicolumn:nnn
          {
            \int_eval:n { #3 -1 }
          }
          { l }
          {
            \exp_not:V \l__grading_scheme_entry_description_tl
          }
        &
        \exp_not:N \__grading_scheme_points:n
          {
            \tl_use:N \l__grading_scheme_entry_points_int
          }
        \\
      }
⟨*log⟩
    \__grading_scheme_log_decr:
    \__grading_scheme_log:x { / Formatting ~ entry ~ '\token_to_str:N #1' }
⟨/log⟩
  }
\cs_new:Npn \__grading_scheme_entry_forrmat:PnnN % implicit #1-4
  {
    \exp_args:NP \__grading_scheme_entry_format:NnnN
  }
```

(*End definition for* `\__grading_scheme_entry_format:NnnN` *and* `\__grading_scheme_entry_format:PnnN`*.*)

__grading_scheme_entry_assert_description:N
__grading_scheme_entry_assert_points:N

These functions check the presence of values of an entry. If an entry is absent, an error message is omitted.

```
\cs_new:Npn \__grading_scheme_entry_assert_description:N #1
  {
    \__grading_scheme_entry_load_description:N #1
    \quark_if_no_value:NT \l__grading_scheme_entry_description_tl
      {
        \msg_error:nnxxx { grading-scheme } { missing-value }
          { entry } { \token_to_str:N #1 } { description }
      }
  }
```

```
\cs_new:Npn \__grading_scheme_entry_assert_points:N #1
  {
    \__grading_scheme_entry_load_points:N #1
    \quark_if_no_value:NT \l__grading_scheme_entry_points_int
      {
        \msg_error:nnxxx { grading-scheme } { missing-value }
          { entry } { \token_to_str:N #1 } { points }
      }
  }
\msg_new:nnn { grading-scheme } { missing-value }
  {
    #1 ~ '#2' ~ has ~ no ~ #3.
  }
```

(*End definition for* `\__grading_scheme_entry_assert_description:N` *and* `\__grading_scheme_entry_assert_points:N`*.*)

4.9 Blocks

4.9.1 Struct setting / reading

A ⟨*block*⟩ is for us the following:

```
struct Block {
  clist* elements;
  tl     text;
  tl     operation;
}
```

`\grading_scheme_block_new:N`
`\grading_scheme_block_clear:N`
`\grading_scheme_block_set_eq:NN`
`\grading_scheme_block_gclear:N`
`\grading_scheme_block_gset_eq:NN`

Do what thear names suggest. We just forward these to the property lists.

```
\cs_set_eq:NN \grading_scheme_block_new:N      \prop_new:N
\cs_set_eq:NN \grading_scheme_block_clear:N    \prop_clear:N
\cs_set_eq:NN \grading_scheme_block_set_eq:NN  \prop_gset_eq:NN
\cs_set_eq:NN \grading_scheme_block_gclear:N   \prop_gclear:N
\cs_set_eq:NN \grading_scheme_block_gset_eq:NN \prop_gset_eq:NN
```

(*End definition for* `\grading_scheme_block_new:N` *and others. These functions are documented on page* **??***.*)

`\l__grading_scheme_block_elements_clist_ptr`
`\l__grading_scheme_block_description_tl`
`\l__grading_scheme_block_operation_tl`

The mentioned local variables where we retrieve values.

```
\ptr_new:N \l__grading_scheme_block_elements_clist_ptr
\tl_new:N  \l__grading_scheme_block_description_tl
\tl_new:N  \l__grading_scheme_block_operation_tl
```

(*End definition for* `\l__grading_scheme_block_elements_clist_ptr`*,* `\l__grading_scheme_block_description_tl`*, and* `\l__grading_scheme_block_operation_tl`*.*)

`\grading_scheme_block_set_description:Nn`
`\grading_scheme_block_gset_description:Nn`
`\grading_scheme_block_set_operation_tl:Nn`
`\grading_scheme_block_gset_operation_tl:Nn`
`\grading_scheme_block_set_elements:NN`
`\grading_scheme_block_gset_elements:NN`
`\grading_scheme_block_set_elements:NP`
`\grading_scheme_block_gset_elements:NP`

Set the description / operation or elements of the block.

When setting elements, a ⟨*clist var*⟩ is expected.

```
\cs_new:Npn \grading_scheme_block_set_description:Nn #1 % implicit description
  {
    \prop_put:Nnn #1 { description }
  }
\cs_new:Npn \grading_scheme_block_gset_description:Nn #1 % implicit description
```

```
  {
    \prop_gput:Nnn #1 { description }
  }
\cs_new:Npn \grading_scheme_block_set_operation:Nn #1 % implicit operation
  {
    \prop_put:Nnn #1 { operation }
  }
\cs_new:Npn \grading_scheme_block_gset_operation:Nn #1 % implicit operation
  {
    \prop_gput:Nnn #1 { operation }
  }
\cs_new:Npn \grading_scheme_block_set_elements:NN #1 % implicit elements clist
  {
    \prop_put:Nnn #1 { elements }
  }
\cs_new:Npn \grading_scheme_block_gset_elements:NN #1 % implicit elements clist
  {
    \prop_gput:Nnn #1 { elements }
  }
\cs_new:Npn \grading_scheme_block_set_elements:NP
  {
    \exp_args:NNP \grading_scheme_block_set_elements:NN
  }
\cs_new:Npn \grading_scheme_block_gset_elements:NP
  {
    \exp_args:NNP \grading_scheme_block_gset_elements:NN
  }
```

(*End definition for* `\grading_scheme_block_set_description:Nn` *and others. These functions are documented on page* **??**.)

\grading_scheme_block_get_description:NN
\grading_scheme_block_get_operation:NN
\grading_scheme_block_get_elements:NN

`\grading_scheme_block_get_description:NN`⟨*block var*⟩⟨*tl var*⟩
`\grading_scheme_block_get_operation:NN`⟨*block var*⟩⟨*tl var*⟩
`\grading_scheme_block_get_elements:NN`⟨*block var*⟩⟨*clist ptr*⟩

Get access to the members of the block. The assignment is local. The returned values might be `\q_no_value` if no value is present.

```
\cs_new:Npn \grading_scheme_block_get_description:NN #1 % implicit #2
  {
    \prop_get:NnN #1 { description }
  }
\cs_new:Npn \grading_scheme_block_get_operation:NN #1 % implicit #2
  {
    \prop_get:NnN #1 { operation }
  }
\cs_new:Npn \grading_scheme_block_get_elements:NN #1 % implicit #2
  {
    \prop_get:NnN #1 { elements }
  }
```

(*End definition for* `\grading_scheme_block_get_description:NN`, `\grading_scheme_block_get_operation:NN`, *and* `\grading_scheme_block_get_elements:NN`. *These functions are documented on page* **??**.)

__grading_scheme_block_load_description:N
__grading_scheme_block_load_operation:N
__grading_scheme_block_load_elements:NN

Loads the members into the corresponding local variables.

```
\cs_new:Npn \__grading_scheme_block_load_description:N #1
```

```
  {
    \grading_scheme_block_get_description:NN #1 \l__grading_scheme_block_description_tl
  }
\cs_new:Npn \__grading_scheme_block_load_operation:N #1
  {
    \grading_scheme_block_get_operation:NN #1 \l__grading_scheme_block_operation_tl
  }
\cs_new:Npn \__grading_scheme_block_load_elements:N #1
  {
    \grading_scheme_block_get_elements:NN #1 \l__grading_scheme_block_elements_clist_ptr
  }
```

(*End definition for* `\__grading_scheme_block_load_description:N`*,* `\__grading_scheme_block_load_operation:N`*, and* `\__grading_scheme_block_load_elements:NN`*.*)

`\__grading_scheme_block_ensure_elements:N`

Ensures that this block has an elements attribute that is a valid clist pointer.

If no value is present, or the pointer is `NULL`, a pointer and/or the clist variable are created.

```
\cs_new:Npn \__grading_scheme_block_ensure_elements:N % implicit #1
  {
    \cs_set_eq:NN
      \__grading_scheme_block_set_elements_optg:NP
      \grading_scheme_block_set_elements:NP
    \__grading_scheme_block_ensure_elements_aux:nN { l }
  }
\cs_new:Npn \__grading_scheme_block_gensure_elements:N % implicit #1
  {
    \cs_set_eq:NN
      \__grading_scheme_block_set_elements_optg:NP
      \grading_scheme_block_gset_elements:NP
    \__grading_scheme_block_ensure_elements_aux:nN { g } %i #1
  }
% prefix, block
% assumes that \__grading_scheme_block_set_elements_optg:NP has been
\cs_new:Npn \__grading_scheme_block_ensure_elements_aux:nN #1 #2
  {
    \__grading_scheme_block_load_elements:N #2
    \quark_if_no_value:NT \l__grading_scheme_block_elements_clist_ptr
      {
        \ptr_clear:Nn \l__grading_scheme_block_elements_clist_ptr { #1 }
        \__grading_scheme_block_set_elements_optg:NP
          #2
          \l__grading_scheme_block_elements_clist_ptr
      }
    \ptr_if_null:NT \l__grading_scheme_block_elements_clist_ptr
      {
        \clist_new:P \l__grading_scheme_block_elements_clist_ptr
      }
  }
```

(*End definition for* `\__grading_scheme_block_ensure_elements:N`*.*)

`\grading_scheme_block_add_element:NN`
`\grading_scheme_block_gadd_element:NN`
`\grading_scheme_block_add_element:PP`
`\grading_scheme_block_gadd_element:PP`
`\grading_scheme_block_add_element:PN`
`\grading_scheme_block_gadd_element:PN`

`\grading_scheme_block_add_element:NN`⟨*block var*⟩⟨*element var*⟩

Add an element to the block.

```
\cs_new:Npn \grading_scheme_block_add_element:NN #1 % implicit element
  {
    \__grading_scheme_block_ensure_elements:N #1 % also loads local variable
    \clist_put_right:Pn \l__grading_scheme_block_elements_clist_ptr
  }
\cs_new:Npn \grading_scheme_block_gadd_element:NN #1 % implicit element
  {
    \__grading_scheme_block_gensure_elements:N #1 % also loads local variable
    \clist_gput_right:Pn \l__grading_scheme_block_elements_clist_ptr
  }
\cs_new:Npn \grading_scheme_block_add_element:PP
  {
    \exp_args:NPP \grading_scheme_block_add_element:NN
  }
\cs_new:Npn \grading_scheme_block_gadd_element:PP
  {
    \exp_args:NPP \grading_scheme_block_gadd_element:NN
  }
\cs_new:Npn \grading_scheme_block_add_element:PN
  {
    \exp_args:NP \grading_scheme_block_add_element:NN
  }
\cs_new:Npn \grading_scheme_block_gadd_element:PN
  {
    \exp_args:NP \grading_scheme_block_gadd_element:NN
  }
```

(*End definition for* `\grading_scheme_block_add_element:NN` *and others. These functions are documented on page* **??**.)

4.9.2 Formatting blocks

`\grading_scheme_block_format:NnnnN`
`\grading_scheme_block_gformat:NnnnN`
`\grading_scheme_block_format:PnnnN`
`\grading_scheme_block_gformat:PnnnN`

`\grading_scheme_block_format:NnnnN`⟨*block var*⟩{⟨*indent*⟩}{⟨*width*⟩}{⟨*indent first row*⟩}⟨*tl var*⟩

Formats this block and puts the control sequence into ⟨*tl var*⟩.

The ⟨*indent*⟩ and ⟨*width*⟩ work as in `\grading_scheme_entry_format:NnnN`.

⟨*indent first row*⟩ can be any ⟨*boolean expression*⟩ and indicates if the first row is also indented. Set this to false to start the block in a row of the tabular that already has contents.

```
\cs_new:Npn \grading_scheme_block_format:NnnnN %implicit #1-5
  {
    \cs_set_eq:NN \__grading_scheme_tl_put_right:Nx \tl_put_right:Nx
    \cs_set_eq:NN \__grading_scheme_element_format_optg:NnnnN \grading_scheme_element_format
    \cs_set_eq:NN \__grading_scheme_tl_set:Nn \tl_set:Nn
    \__grading_scheme_block_format:NnnnN
  }
\cs_new:Npn \grading_scheme_block_gformat:NnnnN %implicit #1-5
  {
    \cs_set_eq:NN \__grading_scheme_tl_put_right:Nx \tl_gput_right:Nx
    \cs_set_eq:NN \__grading_scheme_element_format_optg:NnnnN \grading_scheme_element_gforma
    \cs_set_eq:NN \__grading_scheme_tl_set:Nn \tl_gset:Nn
    \__grading_scheme_block_format:NnnnN
  }
\cs_new:Npn \grading_scheme_block_format:PnnnN % implicit #1-5
```

```
  {
    \exp_args:NP
    \grading_scheme_block_format:NnnnN
  }
\cs_new:Npn \grading_scheme_block_gformat:PnnnN % implicit #1-5
  {
    \exp_args:NP
    \grading_scheme_block_gformat:NnnnN
  }
```

(*End definition for* `\grading_scheme_block_format:NnnnN` *and others. These functions are documented on page* **??**.)

For formatting the block, we need some local variables:

`\l__grading_scheme_block_indent_bool`

Controls whether the `\__grading_scheme_block_indent:nN` macro will actually perform an indent.

```
\bool_new:N \l__grading_scheme_block_indent_bool
```

(*End definition for* `\l__grading_scheme_block_indent_bool`.)

`\__grading_scheme_block_height_int`

Locally stores the height of the block to be typeset.

```
\int_new:N \l__grading_scheme_block_height_int
```

(*End definition for* `\__grading_scheme_block_height_int`.)

`\__grading_scheme_block_format:NnnnN`
`\__grading_scheme_block_format:PnnnN`

Aux function. Assumes that `\__grading_scheme__tl_put_right:Nx` has been set properly.

```
\cs_new:Npn \__grading_scheme_block_format:NnnnN #1 #2 #3 #4 #5
  {
⟨*log⟩
    \__grading_scheme_log:x
      {
        Formatting ~ block ~ '\token_to_str:N #1' ~ into ~ '\token_to_str:N #5'
        ~ with ~ indent ='#2', ~ width ~ '#3' ~ and ~ first ~
        row = '\bool_to_str:n { #4 }'
      }
    \prop_log:N #1
    \__grading_scheme_block_load_elements:N #1
    \exp_args:NP \clist_log:N \l__grading_scheme_block_elements_clist_ptr
    \__grading_scheme_log_incr:
⟨/log⟩
```

We need grouping here so that our indentation boolean is not messed up by recursive calls:

```
    \bool_set:Nn \l__grading_scheme_block_indent_bool { #4 && \c_true_bool }
    \group_begin:
    \__grading_scheme_block_load_description:N #1
    \grading_scheme_block_get_height:NN #1 \l__grading_scheme_block_height_int
```

We now format the description of the block if a value is present.

```
    \quark_if_no_value:NF \l__grading_scheme_block_description_tl
      {
        \__grading_scheme_block_indent:nN { #2 } #5
        \__grading_scheme_tl_put_right:Nx #5
          {
            \exp_not:N \__grading_scheme_multicolumn:nnn
              {
                \int_eval:n { #3 }
              }
              { l| }
              {
                \exp_not:V \l__grading_scheme_block_description_tl
              }
            \\
            \exp_not:N \__grading_scheme_cline:nn
              {
                \int_eval:n { #2 + 2 }
              }
              {
                \int_eval:n { #2 + #3 }
              }
          }
      }
```

Now, we have to format the operation of this block. This is a multirow with rotated description

```
    \__grading_scheme_block_indent:nN { #2 } #5
    \__grading_scheme_block_assert_operation:N #1
    \__grading_scheme_tl_put_right:Nx #5
      {
        \exp_not:N \__grading_scheme_multirow:nnn
          {
            \int_eval:n { \int_use:N \l__grading_scheme_block_height_int - 1 }
          }
          { * }
          {
            \exp_not:N \__grading_scheme_rotatebox:nnn
              { origin = c }
              { 90 }
              {
                \exp_not:N \__grading_scheme_operation:n
                   {
                    \exp_not:V \l__grading_scheme_block_operation_tl
                  }
              }
          }
        &
      }
```

The first element of our block must not be indented:

```
    \bool_set_false:N \l__grading_scheme_block_indent_bool
```

Now, we want to recursively typeset all elements of this block. We need a customized function for this to map over the elements.

```
    \cs_set:Npn \__grading_scheme_block_format_element:N ##1
    {
      \__grading_scheme_element_format_optg:NnnnN
        ##1
        { #2 + 1 }  % indent is one more that the current block
        { #3 - 1 }  % width is one less than current block
        { \l__grading_scheme_block_indent_bool }
        #5
```

This ensures that the second and all further elements will get indented:

```
      \bool_set_true:N \l__grading_scheme_block_indent_bool
    }
    \__grading_scheme_block_ensure_elements:N #1 % load + sanitize elements
    \clist_map_function:PN
      \l__grading_scheme_block_elements_clist_ptr
      \__grading_scheme_block_format_element:N
```

Now, we need to 'smuggle out' the output token list of the current group.

```
    \exp_args:NNNV
      \group_end:
      \__grading_scheme_tl_set:Nn
      #5
      #5
⟨*log⟩
    \__grading_scheme_log_decr:
    \__grading_scheme_log:x { / Formatting ~ block ~ '\token_to_str:N #1' }
⟨/log⟩
  }
\cs_new:Npn \__grading_scheme_block_format:PnnnN
  {
    \exp_args:NP \__grading_scheme_block_format:NnnnN
  }
```

(*End definition for* `\__grading_scheme_block_format:NnnnN` *and* `\__grading_scheme_block_format:PnnnN`*.*)

`\__grading_scheme_block_assert_operation:N`

Asserts that the block has an operation and loads it. If no operation is set, an error is emitted.

```
\cs_new:Npn \__grading_scheme_block_assert_operation:N #1
  {
    \__grading_scheme_block_load_operation:N #1
    \quark_if_no_value:NT \l__grading_scheme_block_operation_tl
      {
         \msg_error:nnxxx { grading-scheme } { missing-value}
           { block }
           { \token_to_str:N #1 }
           { operation }
      }
  }
```

(*End definition for* `\__grading_scheme_block_assert_operation:N`*.*)

`\__grading_scheme_block_indent:nN`

`\__grading_scheme_block_indent:nN{`⟨*indent*⟩`}`⟨*tl var*⟩

Performs the indent into ⟨*tl var*⟩ iff `\l__grading_scheme_block_indent_bool` is true.

Sets `\l__grading_scheme_block_indent_bool` to true afterwards.

```
\cs_new:Npn \__grading_scheme_block_indent:nN #1 #2
  {
    \bool_if:NT \l__grading_scheme_block_indent_bool
      {
        \__grading_scheme_tl_put_right:Nx #2
```

```
     {
      \prg_replicate:nn { #1 } { & }
     }
   }
  \bool_set_true:N \l__grading_scheme_block_indent_bool
 }
```

(*End definition for* `\__grading_scheme_block_indent:nN`.)

4.9.3 Width and height of a block

`\l__grading_scheme_height_sum_int`
`\l__grading_scheme_height_acc_int`

Some temporary int values

```
\int_new:N \l__grading_scheme_height_sum_int
\int_new:N \l__grading_scheme_height_acc_int
```

(*End definition for* `\l__grading_scheme_height_sum_int` *and* `\l__grading_scheme_height_acc_int`.)

`\grading_scheme_block_get_height:NN`
`\grading_scheme_block_get_height:PN`

`\grading_scheme_block_get_height:NN`⟨*block var*⟩⟨*int var*⟩

Gets the height of a block and stores it into the ⟨*int var*⟩.

```
\cs_new:Npn \grading_scheme_block_get_height:NN #1 #2
{
⟨*log⟩
 \__grading_scheme_log:x { Getting ~ height ~ of ~ block ~ '\token_to_str:N #1' }
 \__grading_scheme_log_incr:
⟨/log⟩
```

Grouping is needed to not mess up local variables in the recursion:

```
 \group_begin:
 \__grading_scheme_block_load_elements:N #1
 \int_zero:N \l__grading_scheme_height_sum_int
 \clist_map_function:PN
  \l__grading_scheme_block_elements_clist_ptr
  \__grading_scheme_block_height_accumulator:N
 \grading_scheme_block_if_description:NT #1
  {
   \int_incr:N \l__grading_scheme_height_sum_int
  }
```

Smuggle out this length at return it to the caller:

```
 \exp_args:NNNV
  \group_end:
  \int_set:Nn #2 \l__grading_scheme_height_sum_int
⟨*log⟩
 \__grading_scheme_log_decr:
 \__grading_scheme_log:x
  {
   / Getting ~ height ~ of ~ block ~ '\token_to_str:N #1':
   '\int_use:N #2'
  }
⟨/log⟩
}
\cs_new:Npn \grading_scheme_block_get_height:PN
 {
  \exp_args:NP \grading_scheme_block_get_height:NN
 }
```

(*End definition for* `\grading_scheme_block_get_height:NN` *and* `\grading_scheme_block_get_height:PN`*. These functions are documented on page* **??**.)

`__grading_scheme_block_height_accumulator:N`

This is mapped on the elements of a block and accumulates the heights.

```
\cs_new:Npn \__grading_scheme_block_height_accumulator:N #1
  {
    \grading_scheme_element_get_height:NN #1 \l__grading_scheme_height_acc_int
    \int_add:Nn \l__grading_scheme_height_sum_int
      { \int_use:N \l__grading_scheme_height_acc_int }
  }
```

(*End definition for* `\__grading_scheme_block_height_accumulator:N`.)

`\grading_scheme_block_if_description:NTF`

Tests if the given ⟨*block var*⟩ has a description set or not. Also loads this description

```
\cs_new:Npn \grading_scheme_block_if_description:NT #1 % implicit #2
  {
    \__grading_scheme_block_load_description:N #1
    \quark_if_no_value:NF \l__grading_scheme_block_description_tl
  }
\cs_new:Npn \grading_scheme_block_if_description:NF #1 % implicit #2
  {
    \__grading_scheme_block_load_description:N #1
    \quark_if_no_value:NT \l__grading_scheme_block_description_tl
  }
\cs_new:Npn \grading_scheme_block_if_description:NTF #1 #2 #3
  {
      \__grading_scheme_block_load_description:N #1
      \quark_if_no_value:NTF \l__grading_scheme_block_description_tl
        { #3 }
        { #2 }
  }
```

(*End definition for* `\grading_scheme_block_if_description:NTF`*. This function is documented on page* **??**.)

`\l__grading_scheme_max_width_int`

(*End definition for* `\l__grading_scheme_max_width_int`.)

`\l__grading_scheme_width_acc_int`
`\l__grading_scheme_max_width_int`

Some temporary int values

```
\int_new:N \l__grading_scheme_max_width_int
\int_new:N \l__grading_scheme_width_acc_int
```

(*End definition for* `\l__grading_scheme_width_acc_int` *and* `\l__grading_scheme_max_width_int`.)

`\grading_scheme_block_get_natural_width:NN`
`\grading_scheme_block_get_natural_width:PN`

Gets the “natural” width of a block, that is: The minimal width of this block so that it can be typeset properly.

```
\cs_new:Npn \grading_scheme_block_get_natural_width:NN #1 #2
  {
⟨*log⟩
     \__grading_scheme_log:x
       { Getting ~ natural ~ width ~ of ~ block ~ '\token_to_str:N #1'. }
     \__grading_scheme_log_incr:
⟨/log⟩
```

```
     \group_begin:
     \__grading_scheme_block_load_elements:N #1
     \int_zero:N \l__grading_scheme_max_width_int
     \clist_map_function:PN
       \l__grading_scheme_block_elements_clist_ptr
       \__grading_scheme_block_width_accumulator:N
     \int_incr:N \l__grading_scheme_max_width_int
     \exp_args:NNNV
       \group_end:
       \int_set:Nn #2 \l__grading_scheme_max_width_int
⟨*log⟩
     \__grading_scheme_log_decr:
     \__grading_scheme_log:x
       { / Getting ~ natural ~ width ~ of ~ block ~ '\token_to_str:N #1'. }
⟨/log⟩
  }
\cs_new:Npn \grading_scheme_block_get_natural_width:PN
  {
    \exp_args:NP \grading_scheme_block_get_natural_width:NN
  }
```

(*End definition for* `\grading_scheme_block_get_natural_width:NN` *and* `\grading_scheme_block_get_natural_width:PN`. *These functions are documented on page* **??**.)

`\__grading_scheme_block_width_accumulator:N` Gets the natural width of subelements and accumulates the maximum of them.

```
\cs_new:Npn \__grading_scheme_block_width_accumulator:N #1
  {
⟨*log⟩
    \__grading_scheme_log:x { Accumulating ~ width ~ of ~ '\token_to_str:N #1' }
    \__grading_scheme_log_incr:
⟨/log⟩
    \grading_scheme_element_get_natural_width:NN #1 \l__grading_scheme_width_acc_int
    \int_set:Nn \l__grading_scheme_max_width_int
      {
        \int_max:nn
          \l__grading_scheme_width_acc_int
          \l__grading_scheme_max_width_int
      }
⟨*log⟩
    \__grading_scheme_log_decr:
    \__grading_scheme_log:x { / Accumulationg ~ width ~ of ~ '\token_to_str:N #1' }
⟨/log⟩
  }
```

(*End definition for* `\__grading_scheme_block_width_accumulator:N`.)

4.10 Elements

As discussed earlier, an ⟨*element*⟩ is either a ⟨*block*⟩ or an ⟨*entry*⟩. Thus, our natural representation is as follows:

```
struct Element {
  void* content;
  tl type;
}
```

which essentially just implements a `union` that knows of its current member.

__grading_scheme_element_new:N
__grading_scheme_element_clear:N
__grading_scheme_element_set_eq:NN
__grading_scheme_element_gclear:N
__grading_scheme_element_gset_eq:NN

Do what these say. Forward to the underlying property lists.

```
\cs_set_eq:NN \grading_scheme_element_new:N      \prop_new:N
\cs_set_eq:NN \grading_scheme_element_clear:N    \prop_clear:N
\cs_set_eq:NN \grading_scheme_element_set_eq:NN  \prop_set_eq:NN
\cs_set_eq:NN \grading_scheme_element_gclear:N   \prop_gclear:N
\cs_set_eq:NN \grading_scheme_element_gset_eq:NN \prop_gset_eq:NN
```

(*End definition for* `\__grading_scheme_element_new:N` *and others.*)

\l__grading_scheme_element_content_void_ptr
\l__grading_scheme_element_type_str

Mentioned local variables.

```
\ptr_new:N \l__grading_scheme_element_content_void_ptr
\str_new:N \l__grading_scheme_element_type_str
```

(*End definition for* `\l__grading_scheme_element_content_void_ptr` *and* `\l__grading_scheme_element_type_str`*.*)

\grading_scheme_element_set_block:NN
\grading_scheme_element_gset_block:NN
\grading_scheme_element_set_entry:NN
\grading_scheme_element_gset_entry:NN

`\grading_scheme_element_set_block:NN` ⟨*element var*⟩⟨*block var*⟩

```
\cs_new:Npn \grading_scheme_element_set_block:NN #1 % implicit #2
  {
    \prop_put:Nnn #1 { type } { block }
    \prop_put:Nnn #1 { content }
  }
\cs_new:Npn \grading_scheme_element_gset_block:NN #1 % implicit #2
  {
    \prop_gput:Nnn #1 { type } { block }
    \prop_gput:Nnn #1 { content }
  }
\cs_new:Npn \grading_scheme_element_set_entry:NN #1 % implicit #2
  {
    \prop_put:Nnn #1 { type } { entry }
    \prop_put:Nnn #1 { content }
  }
\cs_new:Npn \grading_scheme_element_gset_entry:NN #1 % implicit #2
  {
    \prop_gput:Nnn #1 { type } { entry }
    \prop_gput:Nnn #1 { content }
  }
```

(*End definition for* `\grading_scheme_element_set_block:NN` *and others. These functions are documented on page* **??***.*)

\grading_scheme_element_get_content:NN
\grading_scheme_element_get_type:NN

`\grading_scheme_element_get_content:NN` ⟨*element var*⟩⟨*void ptr*⟩
`\grading_scheme_element_get_type:NN` ⟨*element var*⟩⟨*str var*⟩

Get the contents. The assignment is local.

```
\cs_new:Npn \grading_scheme_element_get_content:NN #1 % implicit #2
  {
    \prop_get:NnN #1 { content }
  }
\cs_new:Npn \grading_scheme_element_get_type:NN #1 % implicit #2
  {
    \prop_get:NnN #1 { type }
  }
```

(*End definition for* `\grading_scheme_element_get_content:NN` *and* `\grading_scheme_element_get_type:NN`*. These functions are documented on page* **??**.)

`\__grading_scheme_element_load_content:N`
`\__grading_scheme_element_load_type:NN`

Loads the element into the local variables.

```
\cs_new:Npn \__grading_scheme_element_load_content:N #1
  {
    \grading_scheme_element_get_content:NN #1 \l__grading_scheme_element_content_void_ptr
  }
\cs_new:Npn \__grading_scheme_element_load_type:N #1
  {
    \grading_scheme_element_get_type:NN #1 \l__grading_scheme_element_type_str
  }
```

(*End definition for* `\__grading_scheme_element_load_content:N` *and* `\__grading_scheme_element_load_type:NN`*.*)

`\grading_scheme_element_cases:Nnn`
`\grading_scheme_element_cases:NnnTF`

`\grading_scheme_element_cases`⟨*element var*⟩`{`⟨*entry code*⟩`}{`⟨*block code*⟩`}`

Distinguishes between the cases of the element type.

Also loads the element type.

```
\cs_new:Npn \grading_scheme_element_cases:Nnn % implicit #1-3
  {
    \cs_set_eq:NN \__grading_scheme_str_case:Vnw \str_case:Vn
    \__grading_scheme_element_cases:Nnnw
  }
\cs_new:Npn \grading_scheme_element_cases:NnnT % implicit #1-4
  {
    \cs_set_eq:NN \__grading_scheme_str_case:Vnw \str_case:VnT
    \__grading_scheme_element_cases:Nnnw
  }
\cs_new:Npn \grading_scheme_element_cases:NnnF % implicit #1-4
  {
    \cs_set_eq:NN \__grading_scheme_str_case:Vnw \str_case:VnF
    \__grading_scheme_element_cases:Nnnw
  }
\cs_new:Npn \grading_scheme_element_cases:NnnTF % implicit #1-5
  {
    \cs_set_eq:NN \__grading_scheme_str_case:Vnw \str_case:VnTF
    \__grading_scheme_element_cases:Nnnw
  }
```

(*End definition for* `\grading_scheme_element_cases:NnnTF`*. This function is documented on page* **??**.)

`\__grading_scheme_element_cases:Nnnw`

This assumes that `\__grading_scheme_str_case:Vnw` has been set to a `\str_case:VnTF` variant and uses this variant for switching the type cases.

```
\cs_new:Npn \__grading_scheme_element_cases:Nnnw #1 #2 #3 % implicit branches
  {
    \__grading_scheme_element_load_type:N #1
    \__grading_scheme_str_case:Vnw \l__grading_scheme_element_type_str
      {
        { entry } { #2 }
        { block } { #3 }
      }
    % implicit branches
  }
```

(*End definition for* `\__grading_scheme_element_cases:Nnnw`*.*)

\grading_scheme_element_format:NnnnN
\grading_scheme_element_gformat:NnnnN
\grading_scheme_element_format:PnnnN
\grading_scheme_element_gformat:PnnnN

Same syntax as `\grading_scheme_block_format:NnnnN`.

```
\cs_new:Npn \grading_scheme_element_format:NnnnN % implicit #1-5
  {
    \cs_set_eq:NN \__grading_scheme_entry_format_aux:PnnN  \grading_scheme_entry_format:PnnN
    \cs_set_eq:NN \__grading_scheme_block_format_aux:PnnnN \grading_scheme_block_format:Pnnn
    \cs_set_eq:NN \__grading_scheme_tl_put_right:Nx        \tl_put_right:Nx
    \__grading_scheme_element_format:NnnnN
  }
\cs_new:Npn \grading_scheme_element_gformat:NnnnN % implicit #1-5
  {
    \cs_set_eq:NN \__grading_scheme_entry_format_aux:PnnN  \grading_scheme_entry_gformat:Pnn
    \cs_set_eq:NN \__grading_scheme_block_format_aux:PnnnN \grading_scheme_block_gformat:Pnn
    \cs_set_eq:NN \__grading_scheme_tl_put_right:Nx        \tl_put_gright:Nx
    \__grading_scheme_element_format:NnnnN
  }
\cs_new:Npn \grading_scheme_element_format:PnnnN % implicit #1-5
  {
    \exp_args:NP \grading_scheme_element_format:NnnnN
  }
\cs_new:Npn \grading_scheme_element_gformat:PnnnN % implicit #1-5
  {
    \exp_args:NP \grading_scheme_element_gformat:NnnnN
  }
```

(*End definition for* `\grading_scheme_element_format:NnnnN` *and others. These functions are documented on page* **??***.*)

__grading_scheme_element_format:NnnnN
__grading_scheme_element_format:PnnnN

Aux function.

```
\cs_new:Npn \__grading_scheme_element_format:NnnnN #1 #2 #3 #4 #5
  {
⟨*log⟩
    \__grading_scheme_log:x
      {
        Formatting ~ element ~ '\token_to_str:N #1' ~ into ~ '\token_to_str:N #5'.
      }
    \__grading_scheme_log_incr:
    \prop_log:N #1
⟨/log⟩
    \__grading_scheme_element_load_content:N #1
    \grading_scheme_element_cases:NnnF #1
        {
          \bool_if:nTF { #4 }
            {
              \__grading_scheme_entry_format_aux:PnnN
                \l__grading_scheme_element_content_void_ptr
                { #2 }
                { #3 }
                #5
            }
            {
              \__grading_scheme_entry_format_aux:PnnN
                \l__grading_scheme_element_content_void_ptr
                { 0 }
                { #3 }
                #5
            }
        }
        {
          \__grading_scheme_block_format_aux:PnnnN
            \l__grading_scheme_element_content_void_ptr
            { #2 }
            { #3 }
            { #4 }
            #5
        }
        {
          \msg_error:nnxxx { grading-scheme } { missing-value }
            { element }
            { \token_to_str:N #1 }
            { type / content }
        }
    \__grading_scheme_tl_put_right:Nx #5
      {
        \exp_not:N \__grading_scheme_cline:nn
          {
            \int_eval:n { #2 +1 }
          }
          {
            \int_eval:n { #2 + #3 }
          }
      }
⟨*log⟩
    \__grading_scheme_log_decr:
    \__grading_scheme_log:x { Done typesetting ~ element ~ '\token_to_str:N #1' }
⟨/log⟩
  }
\cs_new:Npn \__grading_scheme_element_format:PnnnN % implicit #1-5
  {
    \exp_args:NP \__grading_scheme_element_format:NnnnN
  }
```

(*End definition for* `\__grading_scheme_element_format:NnnnN` *and* `\__grading_scheme_element_format:PnnnN`*.*)

`\grading_scheme_element_get_height:NN`
`\grading_scheme_element_get_height:PN`

Get the the height of an element.

```
\cs_new:Npn \grading_scheme_element_get_height:NN #1 #2
  {
    \grading_scheme_element_cases:NnnF #1
        {
          \int_set:Nn #2 { 1 }
        }
        {
          \__grading_scheme_element_load_content:N #1
          \grading_scheme_block_get_height:PN
            \l__grading_scheme_element_content_void_ptr
            #2
        }
        {
          \msg_error:nnxxx { grading-scheme } { missing-value }
            { element }
            { \token_to_str:N #1 }
            { type / content }
        }
  }
\cs_new:Npn \grading_scheme_element_get_height:PN
  {
    \exp_args:NP \grading_scheme_element_get_height:NN
  }
```

(*End definition for* `\grading_scheme_element_get_height:NN` *and* `\grading_scheme_element_get_height:PN`*. These functions are documented on page* **??**.)

grading_scheme_element_get_natural_width:NN
grading_scheme_element_get_natural_width:PN

Get the natural width of an element

```
\cs_new:Npn \grading_scheme_element_get_natural_width:NN #1 #2
  {
    \grading_scheme_element_cases:NnnF #1
      {
        \int_set:Nn { #2 } { 2 }
      }
      {
        \__grading_scheme_element_load_content:N #1
        \grading_scheme_block_get_natural_width:PN
          \l__grading_scheme_element_content_void_ptr
          #2
      }
      {
        \msg_error:nnxxx { grading-scheme } { missing-value }
          { element }
          { \token_to_str:N #1 }
          { type / content }
      }
  }
\cs_new:Npn \grading_scheme_element_get_natural_width:PN
  {
    \exp_args:NP \grading_scheme_element_get_natural_width:NN
  }
```

(*End definition for* `\grading_scheme_element_get_natural_width:NN` *and* `\grading_scheme_element_get_natural_width:PN`*. These functions are documented on page* **??**.)

`\l__grading_scheme_width_int`

Local int vars for typesetting.

```
\int_new:N \l__grading_scheme_width_int
```

(*End definition for* `\l__grading_scheme_width_int`*.*)

`\g__grading_scheme_table_tl`

Token list where we will build the table.

```
\tl_new:N \g__grading_scheme_table_tl
```

(*End definition for* `\g__grading_scheme_table_tl`*.*)

\grading_scheme_element_typeset:N
\grading_scheme_element_typeset:P

Typesets this element as a tabular and inserts this into the output stream.

```
\cs_new:Npn \grading_scheme_element_typeset:N #1
  {
    \grading_scheme_element_get_natural_width:NN #1 \l__grading_scheme_width_int
    \tl_gclear:N \g__grading_scheme_table_tl
    \tl_gput_right:Nn \g__grading_scheme_table_tl
      {
        \begin{tabular}
      }
    \tl_gput_right:Nx \g__grading_scheme_table_tl
      {
        {
          \prg_replicate:nn { \l__grading_scheme_width_int -1 } { |l }
          | l |
        }
        \exp_not:N \__grading_scheme_hline:
      }
    \grading_scheme_element_gformat:NnnnN
      #1
      { 0 }
      { \l__grading_scheme_width_int }
      { \c_false_bool }
      \g__grading_scheme_table_tl
    \tl_gput_right:Nn \g__grading_scheme_table_tl
      {
        \end{tabular}
      }
    \group_begin:
    \tl_set:Nn \arraystretch { 1.5 }
    \tl_use:N \g__grading_scheme_table_tl
    \group_end:
  }
\cs_new:Npn \grading_scheme_element_typeset:P
  {
    \exp_args:NP \grading_scheme_element_typeset:N
  }
```

(*End definition for* `\grading_scheme_element_typeset:N` *and* `\grading_scheme_element_typeset:P`. *These functions are documented on page* **??**.)

4.11 Facilities for populating data structures

TODO: what about global / local assignments here?

\grading_scheme_entry_new:Nnn
\grading_scheme_entry_new:Pnn

`\grading_scheme_entry_new:Nnn⟨entry var⟩{⟨description⟩}{⟨points⟩}`

```
\cs_new:Npn \grading_scheme_entry_new:Nnn #1 #2 % implcit #3
  {
    \grading_scheme_entry_new:N #1
    \grading_scheme_entry_gset_description:Nn #1 { #2 }
    \grading_scheme_entry_gset_points:Nn #1
  }
\cs_new:Npn \grading_scheme_entry_new:Pnn
  {
    \exp_args:NP \grading_scheme_entry_new:Nnn
  }
```

(*End definition for* \grading_scheme_entry_new:Nnn *and* \grading_scheme_entry_new:Pnn*. These functions are documented on page* **??**.)

\grading_scheme_block_new:Nnn
\grading_scheme_block_new:Pnn

\grading_scheme_block_new:Nnn⟨*block var*⟩{⟨*description*⟩}{⟨*operation*⟩}

```
\cs_new:Npn \grading_scheme_block_new:Nnn #1 #2 % implicit #3
  {
    \grading_scheme_block_new:N #1
    \grading_scheme_block_gset_description:Nn #1 { #2 }
    \grading_scheme_block_gset_operation:Nn #1
  }
\cs_new:Npn \grading_scheme_block_new:Pnn
  {
    \exp_args:NP \grading_scheme_block_new:Nnn
  }
```

(*End definition for* \grading_scheme_block_new:Nnn *and* \grading_scheme_block_new:Pnn*. These functions are documented on page* **??**.)

\grading_scheme_block_new:Nn
\grading_scheme_block_new:Pn

\grading_scheme_block_new:Nn⟨*block var*⟩{⟨*operation*⟩}
Same as above, but no description provided.

```
\cs_new:Npn \grading_scheme_block_new:Nn #1 % implicit #2
  {
    \grading_scheme_block_new:N #1
    \grading_scheme_block_gset_operation:Nn #1
  }
\cs_new:Npn \grading_scheme_block_new:Pn
  {
    \exp_args:NP \grading_scheme_block_new:Nn
  }
```

(*End definition for* \grading_scheme_block_new:Nn *and* \grading_scheme_block_new:Pn*. These functions are documented on page* **??**.)

\grading_scheme_element_from_entry_new:NN
\grading_scheme_element_from_entry_new:NP
\grading_scheme_element_from_entry_new:PP
\grading_scheme_element_gfrom_entry_new:NN
\grading_scheme_element_gfrom_entry_new:NP
\grading_scheme_element_gfrom_entry_new:PP

\grading_scheme_element_from_entry_new:NN⟨*element var*⟩⟨*entry var*⟩
wraps the entry into a new element.

```
\cs_new:Npn \grading_scheme_element_from_entry_new:NN #1 #2
  {
    \grading_scheme_element_new:N #1
    \grading_scheme_element_set_entry:NN #1 #2
  }
\cs_new:Npn \grading_scheme_element_from_entry_new:NP
  {
    \exp_args:NP \grading_scheme_element_from_entry_new:NN
  }
\cs_new:Npn \grading_scheme_element_from_entry_new:PP
  {
    \exp_args:NPP \grading_scheme_element_from_entry_new:NN
  }
\cs_new:Npn \grading_scheme_element_gfrom_entry_new:NN #1 #2
  {
    \grading_scheme_element_new:N #1
    \grading_scheme_element_gset_entry:NN #1 #2
  }
\cs_new:Npn \grading_scheme_element_gfrom_entry_new:NP
  {
    \exp_args:NP \grading_scheme_element_gfrom_entry_new:NN
  }
\cs_new:Npn \grading_scheme_element_gfrom_entry_new:PP
  {
    \exp_args:NPP \grading_scheme_element_gfrom_entry_new:NN
  }
```

(*End definition for* `\grading_scheme_element_from_entry_new:NN` *and others. These functions are documented on page* **??**.)

`\grading_scheme_element_from_block_new:NN`
`\grading_scheme_element_from_block_new:NP`
`\grading_scheme_element_from_block_new:NP`
`\grading_scheme_element_gfrom_block_new:NN`
`\grading_scheme_element_from_gblock_new:NP`
`\grading_scheme_element_gfrom_block_new:NP`

`\grading_scheme_element_from_block_new:NN`⟨*element var*⟩⟨*block var*⟩ wraps the block into a new element.

```
\cs_new:Npn \grading_scheme_element_from_block_new:NN #1 #2
  {
    \grading_scheme_element_new:N #1
    \grading_scheme_element_set_block:NN #1 #2
  }
\cs_new:Npn \grading_scheme_element_from_block_new:NP
  {
    \exp_args:NP \grading_scheme_element_from_block_new:NN
  }
\cs_new:Npn \grading_scheme_element_from_block_new:PP
  {
    \exp_args:NPP \grading_scheme_element_from_block_new:NN
  }
\cs_new:Npn \grading_scheme_element_gfrom_block_new:NN #1 #2
  {
    \grading_scheme_element_new:N #1
    \grading_scheme_element_gset_block:NN #1 #2
  }
\cs_new:Npn \grading_scheme_element_gfrom_block_new:NP
  {
    \exp_args:NP \grading_scheme_element_gfrom_block_new:NN
  }
\cs_new:Npn \grading_scheme_element_gfrom_block_new:PP
  {
    \exp_args:NPP \grading_scheme_element_gfrom_block_new:NN
  }
```

(*End definition for* `\grading_scheme_element_from_block_new:NN` *and others. These functions are documented on page* **??**.)

4.12 Grading scheme environment

When building a grading scheme, we introduce a `\g__grading_scheme_curr_block_ptr` that is a pointer to the current block that is being populated.

Block environments and entries will create new blocks/entries and add themselves to the block pointed out by `\g__grading_scheme_curr_block_ptr`

\l__grading_scheme_entry_ptr
\l__grading_scheme_element_ptr
\l__grading_scheme_curr_block_ptr

```
\ptr_new:Nn \l__grading_scheme_entry_ptr { g }
\ptr_new:Nn \l__grading_scheme_element_ptr { g }
\ptr_new:Nn \l__grading_scheme_curr_block_ptr { g }
```

(*End definition for* `\l__grading_scheme_entry_ptr`*,* `\l__grading_scheme_element_ptr`*, and* `\l__grading_scheme_curr_block_ptr`*.*)

__grading_scheme_entry:nn

Creates an entry in the current block.

```
\cs_new:Npn \__grading_scheme_entry:nn #1 #2
  {
    \ptr_clear:Nn \l__grading_scheme_entry_ptr { g }
    \ptr_clear:Nn \l__grading_scheme_element_ptr { g }
    \grading_scheme_entry_new:Pnn
      \l__grading_scheme_entry_ptr
      { #1 }
      { #2 }
    \grading_scheme_element_gfrom_entry_new:PP
      \l__grading_scheme_element_ptr
      \l__grading_scheme_entry_ptr
    \grading_scheme_block_gadd_element:PP
      \l__grading_scheme_curr_block_ptr
      \l__grading_scheme_element_ptr
  }
```

(*End definition for* `\__grading_scheme_entry:nn`*.*)

__grading_scheme_block_begin:nn
__grading_scheme_block_begin:n
__grading_scheme_block_end:

```
\cs_new:Npn \__grading_scheme_block_begin:nn % implicit #1, #2
  {
    \ptr_clear:Nn \l__grading_scheme_curr_block_ptr { g }
    \grading_scheme_block_new:Pnn \l__grading_scheme_curr_block_ptr
  }
\cs_new:Npn \__grading_scheme_block_begin:n % implicit #1
  {
      \ptr_clear:Nn \l__grading_scheme_curr_block_ptr { g }
      \grading_scheme_block_new:Pn \l__grading_scheme_curr_block_ptr
  }
\cs_new:Npn \__grading_scheme_block_end:
  {
    \ptr_clear:Nn \l__grading_scheme_element_ptr { g }
    \grading_scheme_element_gfrom_block_new:PP
      \l__grading_scheme_element_ptr
      \l__grading_scheme_curr_block_ptr
    \cs_gset:Npx \__grading_scheme_after_group:
      {
        \exp_not:N \grading_scheme_block_gadd_element:PN
          \exp_not:N \l__grading_scheme_curr_block_ptr
          \exp_not:P \l__grading_scheme_element_ptr
      }
    \group_insert_after:N \__grading_scheme_after_group:
  }
```

(*End definition for* `\__grading_scheme_block_begin:nn`*,* `\__grading_scheme_block_begin:n`*, and* `\__grading_scheme_block_end:`*.*)

\entry

```
\NewDocumentCommand \entry { m m }
  {
    \__grading_scheme_entry:nn { #1 } { #2 }
  }
```

(*End definition for* `\entry`*. This function is documented on page* 4*.*)

block

```
\NewDocumentEnvironment { block } { o m }
  {
    \IfValueTF { #1 }
      {
        \__grading_scheme_block_begin:nn { #1 } { #2 }
      }
      {
        \__grading_scheme_block_begin:n { #2 }
      }
  }
  {
    \__grading_scheme_block_end:
  }
```

(*End definition for* `block`*. This function is documented on page* ??*.*)

gradingscheme

```
\NewDocumentEnvironment {gradingscheme} { o m }
  {
    \bool_if:NT \g__grading_scheme_pipe_syntax_bool
      {
        \char_set_active_eq:NN | \__grading_scheme_entry_oneline:w
        \char_set_catcode_active:N |
      }
    \IfValueTF { #1 }
      {
        \__grading_scheme_block_begin:nn { #1 } { #2 }
      }
      {
        \__grading_scheme_block_begin:n { #2 }
      }
  }
  {
    \ptr_clear:Nn \l__grading_scheme_element_ptr { g }
    \grading_scheme_element_gfrom_block_new:PP
      \l__grading_scheme_element_ptr
      \l__grading_scheme_curr_block_ptr
    \grading_scheme_element_typeset:P \l__grading_scheme_element_ptr
  }
```

(*End definition for* `gradingscheme`*. This function is documented on page* ??*.*)

4.13 Implementing the pipe syntax

Everything left is conditional on the pipe option having been specified:

```
\bool_if:NF \g__grading_scheme_pipe_syntax_bool
  {
    \endinput
  }
```

In order to offer syntax based on |, we make | an active character that will read until the end of the line, split at the `@` character, and pass this on to a `\__grading_scheme_entry:nn`

`\s__grading_scheme_endline`

A scan mark that will denote the end of the line for us.

```
\scan_new:N \s__grading_scheme_endline
```

(*End definition for* `\s__grading_scheme_endline`*.*)

`\__grading_scheme_parse_entry:w`

`\__grading_scheme_parse_entry:w` ⟨*description*⟩ `&` ⟨*points*⟩ `\s_endline`

Creates a new entry based on our scan mark.

```
\cs_new:Npn \__grading_scheme_parse_entry:w #1 & #2 \s_endline
  {
    \__grading_scheme_entry:nn { #1 } { #2  }
  }
```

(*End definition for* `\__grading_scheme_parse_entry:w`*.*)

`\__grading_scheme_replace_newline:w`
`\__grading_scheme_replace_newline_aux:w`

Calling `\__grading_scheme_replace_newline:w` replaces the next occurence of a newline character with `\s__grading_scheme_endline`.

This is done by briefly switching category codes.

```
\cs_new:Npn \__grading_scheme_replace_newline:w
  {
    \group_begin:
    \char_set_catcode_active:N\^^M
    \__grading_scheme_replace_newline_aux:w
  }
```

We need to briefly make the newline character an active character in this source file so that pattern matching works correctly.

```
\char_set_catcode_active:N\^^M
\cs_new:Npn \__grading_scheme_replace_newline_aux:w #1 ^^M
  {
    \group_end:
    #1 \s_endline
  }
\char_set_catcode_end_line:N\^^M
```

(*End definition for* `\__grading_scheme_replace_newline:w` *and* `\__grading_scheme_replace_newline_aux:w`*.*)

`\__grading_scheme_entry_oneline:w`

Parses a one-line entry.

```
\cs_new:Npn \__grading_scheme_entry_oneline:w
  {
    \__grading_scheme_replace_newline:w
    \__grading_scheme_parse_entry:w
  }
```

(*End definition for* `\__grading_scheme_entry_oneline:w`.)

1215 ⟨/package⟩

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.