The l3calc package*
Infix notation arithmetic in LaTeX3

The LaTeX3 Project†

2010/02/07

This is pretty much a straight adaption of the calc package and as such has same syntax for the ⟨*calc expression*⟩. However, there are some noticeable differences.

- The calc expression is expanded fully, which means there are no problems with unfinished conditionals. However, the contents of `\widthof` etc. is not expanded at all. This includes uses in traditional LaTeX as in the array package, which tries to do an `\edef` several times. The code used in l3calc provides self-protection for these cases.
- Muskip registers are supported although they can only be used in `\ratio` if already evaluating a muskip expression. For the other three register types, you can use points.
- All results are rounded, not truncated. More precisely, the primitive TeX operations `\divide` and `\multiply` are not used. The only instance where one will observe an effect is when dividing integers.

This version of l3calc is a now a complete replacement for the original calc package providing the same functionality and will prevent the original calc package from loading.

*This file has version number 1776, last revised 2010/02/07.

†Frank Mittelbach, Denys Duchier, Chris Rowley, Rainer Schöpf, Johannes Braams, Michael Downes, David Carlisle, Alan Jeffrey, Morten Høgholm, Thomas Lotze, Javier Bezos, Will Robertson, Joseph Wright

1 User functions

\maxof
\minof
\widthof
\heightof
\depthof
\totalheightof
\ratio
\real
\setlength
\gsetlength
\addtolength
\gaddtolength
\setcounter
\addtocounter
\stepcounter

See documentation for LaTeX 2_ε package calc.

\calc_maxof:nn
\calc_minof:nn
\calc_widthof:n
\calc_heightof:n
\calc_depthof:n
\calc_totalheightof:n
\calc_ratio:nn
\calc_real:n
\calc_setcounter:nn
\calc_addtocounter:nn
\calc_stepcounter:n

Equivalent commands as the above in the expl3 namespace.

\calc_int_set:Nn
\calc_int_gset:Nn
\calc_int_add:Nn
\calc_int_gadd:Nn
\calc_int_sub:Nn
\calc_int_gsub:Nn

`\calc_int_set:Nn` ⟨*int*⟩ `{`⟨*calc expression*⟩`}`
Evaluates ⟨*calc expression*⟩ and either adds or subtracts it from ⟨*int*⟩ or sets ⟨*int*⟩ to it. These operations can also be global.

\calc_dim_set:Nn
\calc_dim_gset:Nn
\calc_dim_add:Nn
\calc_dim_gadd:Nn
\calc_dim_sub:Nn
\calc_dim_gsub:Nn

`\calc_dim_set:Nn` ⟨*dim*⟩ `{`⟨*calc expression*⟩`}`

Evaluates ⟨*calc expression*⟩ and either adds or subtracts it from ⟨*dim*⟩ or sets ⟨*dim*⟩ to it. These operations can also be global.

\calc_skip_set:Nn
\calc_skip_gset:Nn
\calc_skip_add:Nn
\calc_skip_gadd:Nn
\calc_skip_sub:Nn
\calc_skip_gsub:Nn

`\calc_skip_set:Nn` ⟨*skip*⟩ `{`⟨*calc expression*⟩`}`

Evaluates ⟨*calc expression*⟩ and either adds or subtracts it from ⟨*skip*⟩ or sets ⟨*skip*⟩ to it. These operations can also be global.

\calc_muskip_set:Nn
\calc_muskip_gset:Nn
\calc_muskip_add:Nn
\calc_muskip_gadd:Nn
\calc_muskip_sub:Nn
\calc_muskip_gsub:Nn

`\calc_muskip_set:Nn` ⟨*muskip*⟩ `{`⟨*calc expression*⟩`}`

Evaluates ⟨*calc expression*⟩ and either adds or subtracts it from ⟨*muskip*⟩ or sets ⟨*muskip*⟩ to it. These operations can also be global.

\calc_calculate_box_size:nnn

`\calc_calculate_box_size:nnn {`⟨*dim-set*⟩`}`
`{`⟨*item* 1⟩ ⟨*item* 2⟩ ... ⟨*item n*⟩`} {`⟨*contents*⟩`}`

Sets ⟨*contents*⟩ in a temporary box `\l_tmpa_box`. Then ⟨*dim-set*⟩ is put in front of a loop that inserts `+`⟨*item*$_i$⟩ in front of `\l_tmpa_box` and this is evaluated. For instance, if we wanted to determine the total height of the text `xyz` and store it in `\l_tmpa_dim`, we would call it as.

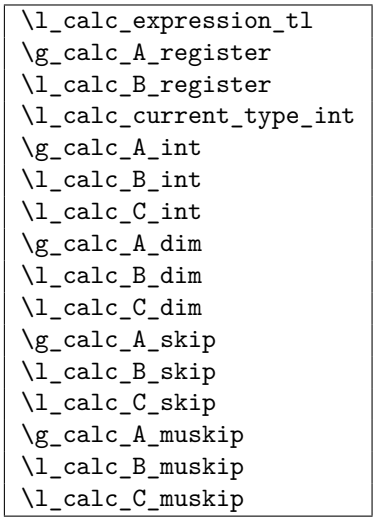
```
\calc_calculate_box_size:nnn
  {\dim_set:Nn\l_tmpa_dim}{\box_ht:N\box_dp:N}{xyz}
```

Similarly, if we wanted the difference between height and depth, we could call it as

```
\calc_calculate_box_size:nnn
  {\dim_set:Nn\l_tmpa_dim}{\box_ht:N{-\box_dp:N}}{xyz}
```

2 l3calc implementation

2.1 Variables



Internal registers.

2.2 Internal functions

```
\calc_assign_generic:NNNNnn
\calc_pre_scan:N
\calc_open:w
\calc_init_B:
\calc_numeric:
\calc_close:
\calc_post_scan:N
\calc_multiply:N
\calc_divide:N
\calc_generic_add_or_subtract:N
\calc_add:
\calc_subtract:
\calc_add_A_to_B:
\calc_subtract_A_from_B:
\calc_generic_multiply_or_divide:N
\calc_multiply_B_by_A:
\calc_divide_B_by_A:
\calc_multiply:
\calc_divide:
\calc_textsize:Nn
\calc_ratio_multiply:nn
\calc_ratio_divide:nn
\calc_real_evaluate:nn
\calc_real_multiply:n
\calc_real_divide:n
\calc_maxmin_operation:Nnn
\calc_maxmin_generic:Nnn
\calc_maxmin_div_or_mul:NNnn
\calc_maxmin_multiply:
\calc_maxmin_multiply:
\calc_error:N
\calc_chk_document_counter:nn
```

Awaiting better documentation :)

2.3 Module code

Since this is basically a re-worked version of the calc package, I haven't bothered with too many comments except for in the places where this package differs. This may (and should) change at some point.

We start by ensuring that the required packages are loaded.

```
1 ⟨*package⟩
2 \ProvidesExplPackage
```

```
  {\filename}{\filedate}{\fileversion}{\filedescription}
\package_check_loaded_expl:
⟨/package⟩
⟨*initex | package⟩
```

\l_calc_expression_tl
\g_calc_A_register
\l_calc_B_register
\l_calc_current_type_int

Here we define some registers and pointers we will need.

```
\tl_new:N \l_calc_expression_tl
\cs_new_nopar:Npn \g_calc_A_register{}
\cs_new_nopar:Npn \l_calc_B_register{}
\int_new:N \l_calc_current_type_int
```

\g_calc_A_int
\l_calc_B_int
\l_calc_C_int
\g_calc_A_dim
\l_calc_B_dim
\l_calc_C_dim
\g_calc_A_skip
\l_calc_B_skip
\l_calc_C_skip
\g_calc_A_muskip
\l_calc_B_muskip
\l_calc_C_muskip

For each type of register we will need three registers to do our manipulations.

```
\int_new:N \g_calc_A_int
\int_new:N \l_calc_B_int
\int_new:N \l_calc_C_int
\dim_new:N \g_calc_A_dim
\dim_new:N \l_calc_B_dim
\dim_new:N \l_calc_C_dim
\skip_new:N \g_calc_A_skip
\skip_new:N \l_calc_B_skip
\skip_new:N \l_calc_C_skip
\muskip_new:N \g_calc_A_muskip
\muskip_new:N \l_calc_B_muskip
\muskip_new:N \l_calc_C_muskip
```

\calc_assign_generic:NNNNnn

The generic function. `#1` is a number denoting which type we are doing. (0=int, 1=dim, 2=skip, 3=muskip), `#2` = temp register A, `#3` = temp register B, `#4` is a function acting on `#5` which is the register to be set. `#6` is the calc expression. We do a little extra work so that `\real` and `\ratio` can still be used by the user.

```
\cs_new:Npn \calc_assign_generic:NNNNnn#1#2#3#4#5#6{
  \cs_set_eq:NN\g_calc_A_register#2
  \cs_set_eq:NN\l_calc_B_register#3
  \int_set:Nn \l_calc_current_type_int {#1}
  \group_begin:
    \cs_set_eq:NN \real \calc_real:n
    \cs_set_eq:NN \ratio\calc_ratio:nn
    \tl_set:Nx\l_calc_expression_tl{#6}
    \exp_after:wN
  \group_end:
  \exp_after:wN\calc_open:w\exp_after:wN(\l_calc_expression_tl !
  \pref_global:D\g_calc_A_register\l_calc_B_register
  \group_end:
  #4{#5}\l_calc_B_register
}
```

A simpler version relying on `\real` and `\ratio` having our definition is

```
\cs_new:Npn \calc_assign_generic:NNNNnn#1#2#3#4#5#6{
  \cs_set_eq:NN\g_calc_A_register#2\cs_set_eq:NN\l_calc_B_register#3
  \int_set:Nn \l_calc_current_type_int {#1}
  \tl_set:Nx\l_calc_expression_tl{#6}
  \exp_after:wN\calc_open:w\exp_after:wN(\l_calc_expression_tl !
  \pref_global:D\g_calc_A_register\l_calc_B_register
  \group_end:
  #4{#5}\l_calc_B_register
}
```

\calc_int_set:Nn
\calc_int_gset:Nn
\calc_int_add:Nn
\calc_int_gadd:Nn
\calc_int_sub:Nn
\calc_int_gsub:Nn

Here are the individual versions for the different register types. First integer registers.

```
\cs_new_nopar:Npn\calc_int_set:Nn{
  \calc_assign_generic:NNNNnn\c_zero\g_calc_A_int\l_calc_B_int\int_set:Nn
}
\cs_new_nopar:Npn\calc_int_gset:Nn{
  \calc_assign_generic:NNNNnn\c_zero\g_calc_A_int\l_calc_B_int\int_gset:Nn
}
\cs_new_nopar:Npn\calc_int_add:Nn{
  \calc_assign_generic:NNNNnn\c_zero\g_calc_A_int\l_calc_B_int\int_add:Nn
}
\cs_new_nopar:Npn\calc_int_gadd:Nn{
  \calc_assign_generic:NNNNnn\c_zero\g_calc_A_int\l_calc_B_int\int_gadd:Nn
}
\cs_new_nopar:Npn\calc_int_sub:Nn{
  \calc_assign_generic:NNNNnn\c_zero\g_calc_A_int\l_calc_B_int\int_sub:Nn
}
\cs_new_nopar:Npn\calc_int_gsub:Nn{
  \calc_assign_generic:NNNNnn\c_zero\g_calc_A_int\l_calc_B_int\int_gsub:Nn
}
```

\calc_dim_set:Nn
\calc_dim_gset:Nn
\calc_dim_add:Nn
\calc_dim_gadd:Nn
\calc_dim_sub:Nn
\calc_dim_gsub:Nn

Dimens.

```
\cs_new_nopar:Npn\calc_dim_set:Nn{
  \calc_assign_generic:NNNNnn\c_one\g_calc_A_dim\l_calc_B_dim\dim_set:Nn
}
\cs_new_nopar:Npn\calc_dim_gset:Nn{
  \calc_assign_generic:NNNNnn\c_one\g_calc_A_dim\l_calc_B_dim\dim_gset:Nn
}
\cs_new_nopar:Npn\calc_dim_add:Nn{
  \calc_assign_generic:NNNNnn\c_one\g_calc_A_dim\l_calc_B_dim\dim_add:Nn
}
\cs_new_nopar:Npn\calc_dim_gadd:Nn{
  \calc_assign_generic:NNNNnn\c_one\g_calc_A_dim\l_calc_B_dim\dim_gadd:Nn
}
\cs_new_nopar:Npn\calc_dim_sub:Nn{
  \calc_assign_generic:NNNNnn\c_one\g_calc_A_int\l_calc_B_int\dim_sub:Nn
}
\cs_new_nopar:Npn\calc_dim_gsub:Nn{
```

```
  \calc_assign_generic:NNNNnn\c_one\g_calc_A_int\l_calc_B_int\dim_gsub:Nn
}
```

\calc_skip_set:Nn
\calc_skip_gset:Nn
\calc_skip_add:Nn
\calc_skip_gadd:Nn
\calc_skip_sub:Nn
\calc_skip_gsub:Nn

Skips.

```
\cs_new_nopar:Npn\calc_skip_set:Nn{
  \calc_assign_generic:NNNNnn\c_two\g_calc_A_skip\l_calc_B_skip\skip_set:Nn
}
\cs_new_nopar:Npn\calc_skip_gset:Nn{
  \calc_assign_generic:NNNNnn\c_two\g_calc_A_skip\l_calc_B_skip\skip_gset:Nn
}
\cs_new_nopar:Npn\calc_skip_add:Nn{
  \calc_assign_generic:NNNNnn\c_two\g_calc_A_skip\l_calc_B_skip\skip_add:Nn
}
\cs_new_nopar:Npn\calc_skip_gadd:Nn{
  \calc_assign_generic:NNNNnn\c_two\g_calc_A_skip\l_calc_B_skip\skip_gadd:Nn
}
\cs_new_nopar:Npn\calc_skip_sub:Nn{
  \calc_assign_generic:NNNNnn\c_two\g_calc_A_skip\l_calc_B_skip\skip_sub:Nn
}
\cs_new_nopar:Npn\calc_skip_gsub:Nn{
  \calc_assign_generic:NNNNnn\c_two\g_calc_A_skip\l_calc_B_skip\skip_gsub:Nn
}
```

\calc_muskip_set:Nn
\calc_muskip_gset:Nn
\calc_muskip_add:Nn
\calc_muskip_gadd:Nn
\calc_muskip_sub:Nn
\calc_muskip_gsub:Nn

Muskips.

```
\cs_new_nopar:Npn\calc_muskip_set:Nn{
  \calc_assign_generic:NNNNnn\c_three\g_calc_A_muskip\l_calc_B_muskip
    \muskip_set:Nn
}
\cs_new_nopar:Npn\calc_muskip_gset:Nn{
  \calc_assign_generic:NNNNnn\c_three\g_calc_A_muskip\l_calc_B_muskip
    \muskip_gset:Nn
}
\cs_new_nopar:Npn\calc_muskip_add:Nn{
  \calc_assign_generic:NNNNnn\c_three\g_calc_A_muskip\l_calc_B_muskip
    \muskip_add:Nn
}
\cs_new_nopar:Npn\calc_muskip_gadd:Nn{
  \calc_assign_generic:NNNNnn\c_three\g_calc_A_muskip\l_calc_B_muskip
    \muskip_gadd:Nn
}
\cs_new_nopar:Npn\calc_muskip_sub:Nn{
  \calc_assign_generic:NNNNnn\c_three\g_calc_A_muskip\l_calc_B_muskip
    \muskip_add:Nn
}
\cs_new_nopar:Npn\calc_muskip_gsub:Nn{
  \calc_assign_generic:NNNNnn\c_three\g_calc_A_muskip\l_calc_B_muskip
    \muskip_gadd:Nn
}
```

\calc_pre_scan:N In case we found one of the special operations, this should just be executed.

```
\cs_new_nopar:Npn \calc_pre_scan:N #1{
  \if_meaning:w(#1
    \exp_after:wN\calc_open:w
  \else:
    \if_meaning:w \calc_textsize:Nn #1
    \else:
      \if_meaning:w \calc_maxmin_operation:Nnn #1
      \else:
```

`\calc_numeric:` uses a primitive assignment so doesn't care about these dangling `\fi:`s.

```
        \calc_numeric:
      \fi:
    \fi:
  \fi:
  #1}
```

\calc_open:w

```
\cs_new_nopar:Npn \calc_open:w({
  \group_begin:\group_execute_after:N\calc_init_B:
  \group_begin:\group_execute_after:N\calc_init_B:
  \calc_pre_scan:N
}
```

\calc_init_B:
\calc_numeric:
\calc_close:

```
\cs_new_nopar:Npn\calc_init_B:{\l_calc_B_register\g_calc_A_register}
\cs_new_nopar:Npn\calc_numeric:{
  \tex_afterassignment:D\calc_post_scan:N
  \pref_global:D\g_calc_A_register
}
\cs_new_nopar:Npn\calc_close:{
  \group_end:\pref_global:D\g_calc_A_register\l_calc_B_register
  \group_end:\pref_global:D\g_calc_A_register\l_calc_B_register
  \calc_post_scan:N}
```

\calc_post_scan:N Look at what token we have and decide where to go.

```
\cs_new_nopar:Npn\calc_post_scan:N#1{
  \if_meaning:w#1!\cs_set_eq:NN\calc_next:w\group_end: \else:
    \if_meaning:w#1+\cs_set_eq:NN\calc_next:w\calc_add: \else:
      \if_meaning:w#1-\cs_set_eq:NN\calc_next:w\calc_subtract: \else:
        \if_meaning:w#1*\cs_set_eq:NN\calc_next:w\calc_multiply:N \else:
          \if_meaning:w#1/\cs_set_eq:NN\calc_next:w\calc_divide:N \else:
            \if_meaning:w#1)\cs_set_eq:NN\calc_next:w\calc_close: \else:
              \if_meaning:w#1\scan_stop:\cs_set_eq:NN\calc_next:w\calc_post_scan:N
              \else:
```

If we get here, there is an error but let's also disable `\calc_next:w` since it is otherwise undefined. No need to give extra errors just for that.

```
                \cs_set_eq:NN \calc_next:w \prg_do_nothing:
                \calc_error:N#1
              \fi:
            \fi:
          \fi:
        \fi:
      \fi:
    \fi:
  \fi:
  \calc_next:w}
```

`\calc_multiply:N` `\calc_divide:N` The switches for multiplication and division.

```
\cs_new_nopar:Npn \calc_multiply:N #1{
  \if_meaning:w \calc_maxmin_operation:Nnn #1
    \cs_set_eq:NN \calc_next:w \calc_maxmin_multiply:
  \else:
    \if_meaning:w \calc_ratio_multiply:nn #1
      \cs_set_eq:NN \calc_next:w \calc_ratio_multiply:nn
    \else:
      \if_meaning:w \calc_real_evaluate:nn #1
        \cs_set_eq:NN \calc_next:w \calc_real_multiply:n
      \else:
        \cs_set_nopar:Npn \calc_next:w{\calc_multiply: #1}
      \fi:
    \fi:
  \fi:
  \calc_next:w
}
\cs_new_nopar:Npn \calc_divide:N #1{
  \if_meaning:w \calc_maxmin_operation:Nnn #1
    \cs_set_eq:NN \calc_next:w \calc_maxmin_divide:
  \else:
    \if_meaning:w \calc_ratio_multiply:nn #1
      \cs_set_eq:NN \calc_next:w \calc_ratio_divide:nn
    \else:
      \if_meaning:w \calc_real_evaluate:nn #1
        \cs_set_eq:NN \calc_next:w \calc_real_divide:n
      \else:
        \cs_set_nopar:Npn \calc_next:w{\calc_divide: #1}
      \fi:
    \fi:
  \fi:
  \calc_next:w
}
```

`\calc_generic_add_or_subtract:N` `\calc_add:` `\calc_subtract:` `\calc_add_A_to_B:` `\calc_subtract_A_from_B:` Here is how we add and subtract.

```
\cs_new_nopar:Npn\calc_generic_add_or_subtract:N#1{
  \group_end:
  \pref_global:D\g_calc_A_register\l_calc_B_register\group_end:
  \group_begin:\group_execute_after:N#1\group_begin:
  \group_execute_after:N\calc_init_B:
  \calc_pre_scan:N}
\cs_new_nopar:Npn\calc_add:{\calc_generic_add_or_subtract:N\calc_add_A_to_B:}
\cs_new_nopar:Npn\calc_subtract:{
  \calc_generic_add_or_subtract:N\calc_subtract_A_from_B:}
```

Don't use `\tex_advance:D` since it allows overflows.

```
\cs_new_nopar:Npn\calc_add_A_to_B:{
  \l_calc_B_register
  \if_case:w\l_calc_current_type_int
  \etex_numexpr:D\or:
  \etex_dimexpr:D\or:
  \etex_glueexpr:D\or:
  \etex_muexpr:D\fi:
  \l_calc_B_register + \g_calc_A_register\scan_stop:
}
\cs_new_nopar:Npn\calc_subtract_A_from_B:{
  \l_calc_B_register
  \if_case:w\l_calc_current_type_int
  \etex_numexpr:D\or:
  \etex_dimexpr:D\or:
  \etex_glueexpr:D\or:
  \etex_muexpr:D\fi:
  \l_calc_B_register - \g_calc_A_register\scan_stop:
}
```

`lc_generic_multiply_or_divide:N`
`\calc_multiply_B_by_A:`
`\calc_divide_B_by_A:`
`\calc_multiply:`
`\calc_divide:`

And here is how we multiply and divide. Note that we do not use the primitive TEX operations but the expandable operations provided by ε-TEX. This means that all results are rounded not truncated!

```
\cs_new_nopar:Npn\calc_generic_multiply_or_divide:N#1{
  \group_end:
  \group_begin:
  \cs_set_eq:NN\g_calc_A_register\g_calc_A_int
  \cs_set_eq:NN\l_calc_B_register\l_calc_B_int
  \int_zero:N \l_calc_current_type_int
  \group_execute_after:N#1\calc_pre_scan:N
}
\cs_new_nopar:Npn\calc_multiply_B_by_A:{
  \l_calc_B_register
  \if_case:w\l_calc_current_type_int
  \etex_numexpr:D\or:
  \etex_dimexpr:D\or:
  \etex_glueexpr:D\or:
  \etex_muexpr:D\fi:
```

```
  \l_calc_B_register*\g_calc_A_int\scan_stop:
}
\cs_new_nopar:Npn\calc_divide_B_by_A:{
  \l_calc_B_register
  \if_case:w\l_calc_current_type_int
  \etex_numexpr:D\or:
  \etex_dimexpr:D\or:
  \etex_glueexpr:D\or:
  \etex_muexpr:D\fi:
  \l_calc_B_register/\g_calc_A_int\scan_stop:
}
\cs_new_nopar:Npn\calc_multiply:{
  \calc_generic_multiply_or_divide:N\calc_multiply_B_by_A:}
\cs_new_nopar:Npn\calc_divide:{
  \calc_generic_multiply_or_divide:N\calc_divide_B_by_A:}
```

`\calc_calculate_box_size:nnn` `\calc_calculate_box_size_aux:n`

Put something in a box and measure it. `#1` is a list of `\box_ht:N` etc., `#2` should be `\dim_set:Nn`⟨*dim register*⟩ or `\dim_gset:Nn`⟨*dim register*⟩ and `#3` is the contents.

```
\cs_new:Npn \calc_calculate_box_size:nnn #1#2#3{
  \hbox_set:Nn \l_tmpa_box {{#3}}
  #2{\c_zero_dim \tl_map_function:nN{#1}\calc_calculate_box_size_aux:n}
}
```

Helper for calculating the final dimension.

```
\cs_set_nopar:Npn \calc_calculate_box_size_aux:n#1{ + #1\l_tmpa_box}
```

`\calc_textsize:Nn`

Now we can define `\calc_textsize:Nn`.

```
\cs_set_protected:Npn \calc_textsize:Nn#1#2{
  \group_begin:
  \cs_set_eq:NN\calc_widthof_aux:n\box_wd:N
  \cs_set_eq:NN\calc_heightof_aux:n\box_ht:N
  \cs_set_eq:NN\calc_depthof_aux:n\box_dp:N
  \cs_set_nopar:Npn\calc_totalheightof_aux:n{\box_ht:N\box_dp:N}
  \exp_args:No\calc_calculate_box_size:nnn{#1}
  {\dim_gset:Nn\g_calc_A_register}
```

Restore the four user commands here since there might be a recursive call.

```
  {
    \cs_set_eq:NN \calc_depthof_aux:n \calc_depthof_auxi:n
    \cs_set_eq:NN \calc_widthof_aux:n \calc_widthof_auxi:n
    \cs_set_eq:NN \calc_heightof_aux:n \calc_heightof_auxi:n
    \cs_set_eq:NN \calc_totalheightof_aux:n \calc_totalheightof_auxi:n
    #2
  }
  \group_end:
  \calc_post_scan:N
}
```

`\calc_ratio_multiply:nn`
`\calc_ratio_divide:nn`

Evaluate a ratio. If we were already evaluation a ⟨*muskip*⟩ register, the ratio is probably also done with this type and we'll have to convert them to regular points.

```
\cs_set_protected:Npn\calc_ratio_multiply:nn#1#2{
  \group_end:\group_begin:
  \if_num:w\l_calc_current_type_int < \c_three
    \calc_dim_set:Nn\l_calc_B_int{#1}
    \calc_dim_set:Nn\l_calc_C_int{#2}
  \else:
    \calc_dim_muskip:Nn{\l_calc_B_int\etex_mutoglue:D}{#1}
    \calc_dim_muskip:Nn{\l_calc_C_int\etex_mutoglue:D}{#2}
  \fi:
```

Then store the ratio as a fraction, which we just pass on.

```
  \cs_gset_nopar:Npx\calc_calculated_ratio:{
    \int_use:N\l_calc_B_int/\int_use:N\l_calc_C_int
  }
  \group_end:
```

Here we set the new value of `\l_calc_B_register` and remember to evaluate it as the correct type. Note that the intermediate calculation is a scaled operation (meaning the intermediate value is 64-bit) so we don't get into trouble when first multiplying by a large number and then dividing.

```
  \l_calc_B_register
  \if_case:w\l_calc_current_type_int
  \etex_numexpr:D\or:
  \etex_dimexpr:D\or:
  \etex_glueexpr:D\or:
  \etex_muexpr:D\fi:
  \l_calc_B_register*\calc_calculated_ratio:\scan_stop:
  \group_begin:
  \calc_post_scan:N}
```

Division is just flipping the arguments around.

```
\cs_new:Npn \calc_ratio_divide:nn#1#2{\calc_ratio_multiply:nn{#2}{#1}}
```

`\calc_real_evaluate:nn`
`\calc_real_multiply:n`
`\calc_real_divide:n`

Although we could define the `\real` function as a subcase of `\ratio`, this is horribly inefficient since we just want to convert the decimal to a fraction.

```
\cs_new_protected_nopar:Npn\calc_real_evaluate:nn #1#2{
  \group_end:
  \l_calc_B_register
  \if_case:w\l_calc_current_type_int
  \etex_numexpr:D\or:
  \etex_dimexpr:D\or:
  \etex_glueexpr:D\or:
  \etex_muexpr:D\fi:
```

```
  \l_calc_B_register *
    \tex_number:D \dim_eval:n{#1pt}/
    \tex_number:D\dim_eval:n{#2pt}
  \scan_stop:
  \group_begin:
  \calc_post_scan:N}
\cs_new_nopar:Npn \calc_real_multiply:n #1{\calc_real_evaluate:nn{#1}{1}}
\cs_new_nopar:Npn \calc_real_divide:n {\calc_real_evaluate:nn{1}}
```

`\calc_maxmin_operation:Nnn`
`\calc_maxmin_generic:Nnn`
`\calc_maxmin_div_or_mul:NNnn`
`\calc_maxmin_multiply:`
`\calc_maxmin_multiply:`

The max and min functions.

```
\cs_set_protected:Npn\calc_maxmin_operation:Nnn#1#2#3{
  \group_begin:
  \calc_maxmin_generic:Nnn#1{#2}{#3}
  \group_end:
  \calc_post_scan:N
}
```

`#1` is either `>` or `<` and was expanded into this initially.

```
\cs_new_protected:Npn \calc_maxmin_generic:Nnn#1#2#3{
  \group_begin:
  \if_case:w\l_calc_current_type_int
    \calc_int_set:Nn\l_calc_C_int{#2}%
    \calc_int_set:Nn\l_calc_B_int{#3}%
    \pref_global:D\g_calc_A_register
    \if_num:w\l_calc_C_int#1\l_calc_B_int
    \l_calc_C_int\else:\l_calc_B_int\fi:
  \or:
    \calc_dim_set:Nn\l_calc_C_dim{#2}%
    \calc_dim_set:Nn\l_calc_B_dim{#3}%
    \pref_global:D\g_calc_A_register
    \if_dim:w\l_calc_C_dim#1\l_calc_B_dim
    \l_calc_C_dim\else:\l_calc_B_dim\fi:
  \or:
    \calc_skip_set:Nn\l_calc_C_skip{#2}%
    \calc_skip_set:Nn\l_calc_B_skip{#3}%
    \pref_global:D\g_calc_A_register
    \if_dim:w\l_calc_C_skip#1\l_calc_B_skip
    \l_calc_C_skip\else:\l_calc_B_skip\fi:
  \else:
    \calc_muskip_set:Nn\l_calc_C_muskip{#2}%
    \calc_muskip_set:Nn\l_calc_B_muskip{#3}%
    \pref_global:D\g_calc_A_register
    \if_dim:w\l_calc_C_muskip#1\l_calc_B_muskip
    \l_calc_C_muskip\else:\l_calc_B_muskip\fi:
  \fi:
  \group_end:
}
\cs_new:Npn\calc_maxmin_div_or_mul:NNnn#1#2#3#4{
```

```
  \group_end:
  \group_begin:
  \int_zero:N\l_calc_current_type_int
  \group_execute_after:N#1
  \calc_maxmin_generic:Nnn#2{#3}{#4}
  \group_end:
  \group_begin:
  \calc_post_scan:N
}
\cs_new_nopar:Npn\calc_maxmin_multiply:{
  \calc_maxmin_div_or_mul:NNnn\calc_multiply_B_by_A:}
\cs_new_nopar:Npn\calc_maxmin_divide:  {
  \calc_maxmin_div_or_mul:NNnn\calc_divide_B_by_A:}
```

`\calc_error:N` The error message.

```
\cs_new_nopar:Npn\calc_error:N#1{
  \PackageError{calc}
  {`\token_to_str:N#1'~ invalid~ at~ this~ point}
  {I~ expected~ to~ see~ one~ of:~ +~ -~ *~ /~ )}
}
```

2.4 Higher level commands

The various operations allowed.

`\calc_maxof:nn` `\calc_minof:nn` `\maxof` `\minof` Max and min operations

```
\cs_new:Npn \calc_maxof:nn#1#2{
  \calc_maxmin_operation:Nnn > \exp_not:n{{#1}{#2}}
}
\cs_new:Npn \calc_minof:nn#1#2{
  \calc_maxmin_operation:Nnn < \exp_not:n{{#1}{#2}}
}
\cs_set_eq:NN \maxof \calc_maxof:nn
\cs_set_eq:NN \minof \calc_minof:nn
```

`\calc_widthof:n` `\calc_widthof_aux:n` `\calc_widthof_auxi:n` `\calc_heightof:n` `\calc_heightof_aux:n` `\calc_heightof_auxi:n` `\calc_depthof:n` `\calc_depthof_aux:n` `\calc_depthof_auxi:n` `\calc_totalheightof:n` `\calc_totalheightof_aux:n` `\calc_totalheightof_auxi:n` Text dimension commands.

```
\cs_new:Npn \calc_widthof:n#1{
  \calc_textsize:Nn \exp_not:N\calc_widthof_aux:n\exp_not:n{{#1}}
}
\cs_new:Npn \calc_heightof:n#1{
  \calc_textsize:Nn \exp_not:N\calc_heightof_aux:n\exp_not:n{{#1}}
}
\cs_new:Npn \calc_depthof:n#1{
  \calc_textsize:Nn \exp_not:N\calc_depthof_aux:n\exp_not:n{{#1}}
}
```

```
\cs_new:Npn \calc_totalheightof:n#1{
  \calc_textsize:Nn \exp_not:N\calc_totalheightof_aux:n \exp_not:n{{#1}}
}
\cs_new:Npn \calc_widthof_aux:n #1{
  \exp_not:N\calc_widthof_aux:n\exp_not:n{{#1}}
}
\cs_new_eq:NN \calc_widthof_auxi:n \calc_widthof_aux:n
\cs_new:Npn \calc_depthof_aux:n #1{
  \exp_not:N\calc_depthof_aux:n\exp_not:n{{#1}}
}
\cs_new_eq:NN \calc_depthof_auxi:n \calc_depthof_aux:n
\cs_new:Npn \calc_heightof_aux:n #1{
  \exp_not:N\calc_heightof_aux:n\exp_not:n{{#1}}
}
\cs_new_eq:NN \calc_heightof_auxi:n \calc_heightof_aux:n
\cs_new:Npn \calc_totalheightof_aux:n #1{
  \exp_not:N\calc_totalheightof_aux:n\exp_not:n{{#1}}
}
\cs_new_eq:NN \calc_totalheightof_auxi:n \calc_totalheightof_aux:n
```

`\calc_ratio:nn` `\calc_real:n`

Ratio and real.

```
\cs_new:Npn \calc_ratio:nn#1#2{
  \calc_ratio_multiply:nn\exp_not:n{{#1}{#2}}}
\cs_new_nopar:Npn \calc_real:n {\calc_real_evaluate:nn}
```

We can implement real and ratio without actually using these names. We'll see.

`\widthof` `\heightof` `\depthof` `\totalheightof` `\ratio` `\real`

User commands.

```
\cs_set_eq:NN \depthof\calc_depthof:n
\cs_set_eq:NN \widthof\calc_widthof:n
\cs_set_eq:NN \heightof\calc_heightof:n
\cs_set_eq:NN \totalheightof\calc_totalheightof:n
%%\cs_set_eq:NN \ratio\calc_ratio:nn
%%\cs_set_eq:NN \real\calc_real:n
```

`\setlength` `\gsetlength` `\addtolength` `\gaddtolength`

```
\cs_set_protected_nopar:Npn \setlength{\calc_skip_set:Nn}
\cs_set_protected_nopar:Npn \gsetlength{\calc_skip_gset:Nn}
\cs_set_protected_nopar:Npn \addtolength{\calc_skip_add:Nn}
\cs_set_protected_nopar:Npn \gaddtolength{\calc_skip_gadd:Nn}
```

`\calc_setcounter:nn` `\calc_addtocounter:nn` `\calc_stepcounter:n` `\setcounter` `\addtocounter` `\stepcounter` `\calc_chk_document_counter:nn`

Document commands for LaTeX 2_ε counters. Also add support for amstext. Note that when l3breqn is used, `\mathchoice` will no longer need this switch as the argument is only executed once.

```
⟨/initex | package⟩
⟨*package⟩
\newif\iffirstchoice@    \firstchoice@true
⟨/package⟩
⟨*initex | package⟩
\cs_set_protected_nopar:Npn \calc_setcounter:nn#1#2{
  \calc_chk_document_counter:nn{#1}{
    \exp_args:Nc\calc_int_gset:Nn {c@#1}{#2}
  }
}
\cs_set_protected_nopar:Npn \calc_addtocounter:nn#1#2{
⟨/initex | package⟩
⟨*package⟩
  \iffirstchoice@
⟨/package⟩
⟨*initex | package⟩
  \calc_chk_document_counter:nn{#1}{
    \exp_args:Nc\calc_int_gadd:Nn {c@#1}{#2}
  }
⟨/initex | package⟩
⟨*package⟩
  \fi:
⟨/package⟩
⟨*initex | package⟩
}
\cs_set_protected_nopar:Npn \calc_stepcounter:n#1{
⟨/initex | package⟩
⟨*package⟩
  \iffirstchoice@
⟨/package⟩
⟨*initex | package⟩
  \calc_chk_document_counter:nn{#1}{
    \int_gincr:c {c@#1}
    \group_begin:
      \cs_set_eq:NN \@elt\@stpelt \use:c{cl@#1}
    \group_end:
  }
⟨/initex | package⟩
⟨*package⟩
  \fi:
⟨/package⟩
⟨*initex | package⟩
}
\cs_new_nopar:Npn \calc_chk_document_counter:nn#1{
  \cs_if_free:cTF{c@#1}{\@nocounterr {#1}}
}
\cs_set_eq:NN \setcounter \calc_setcounter:nn
\cs_set_eq:NN \addtocounter \calc_addtocounter:nn
\cs_set_eq:NN \stepcounter \calc_stepcounter:n
⟨/initex | package⟩
```

```
⟨*package⟩
\AtBeginDocument{
  \cs_set_eq:NN \setcounter \calc_setcounter:nn
  \cs_set_eq:NN \addtocounter \calc_addtocounter:nn
  \cs_set_eq:NN \stepcounter \calc_stepcounter:n
}
```

Prevent the usual calc from loading.

```
\cs_set_nopar:cpn{ver@calc.sty}{2005/08/06}
⟨/package⟩

⟨*showmemory⟩
\showMemUsage
⟨/showmemory⟩
```