LaTeX2 Functional Interfaces for LaTeX3 Programming Layer

Jianrui Lyu (tolvjr@163.com)
https://github.com/lvjr/functional

Version 2022B (2022-03-19)

LaTeX3 programming layer (expl3) is very powerful for advanced users, but it is a little complicated for normal users. This functional package aims to provide intuitive LaTeX2 functional interfaces for it.

Although there are functions in LaTeX3, the evaluation of them is from outside to inside. With this package, the evaluation of functions is from inside to outside, which is the same as other programming languages such as JavaScript or Lua. In this way, it is rather easy to debug code too.

Note that many paragraphs in this manual are copied from the documentation of expl3.
Contents

1 Overview of Features .............................................. 5
  1.1 Evaluation from Inside to Outside .................................. 5
  1.2 Group Scoping of Functions ....................................... 5
  1.3 Tracing Evaluation of Functions .................................... 6
  1.4 Definitions of Functions ......................................... 7
  1.5 Variants of Arguments ........................................... 7

2 Functional Programming (Prg) ...................................... 9
  2.1 Defining Functions and Conditionals .............................. 9
  2.2 Collecting Returned Values ....................................... 9

3 Argument Using (Use) ................................................ 10
  3.1 Expanding Tokens ................................................ 10
  3.2 Using Tokens .................................................... 10

4 Control Structures (Bool) ............................................ 12
  4.1 Constant and Scratch Booleans .................................... 12
  4.2 Creating and Setting Booleans .................................... 12
  4.3 Viewing Booleans ................................................ 13
  4.4 Booleans and Conditionals ....................................... 13

5 Token Lists (Tl) ...................................................... 15
  5.1 Constant and Scratch Token Lists ................................. 15
  5.2 Creating and Using Token Lists .................................. 15
  5.3 Viewing Token Lists ............................................. 16
  5.4 Setting Token List Variables ..................................... 16
  5.5 Replacing Tokens ................................................ 17
  5.6 Working with the Content of Token Lists ......................... 18
  5.7 Mapping over Token Lists ....................................... 19
  5.8 Token List Conditionals ........................................ 20
  5.9 Token List Case Functions ...................................... 21

6 Strings (Str) ........................................................ 23
  6.1 Constant and Scratch Strings .................................... 23
  6.2 Creating and Using Strings ...................................... 23
  6.3 Viewing Strings .................................................. 24
  6.4 Setting String Variables ........................................ 24
6.5 Modifying String Variables ........................................ 25
6.6 Working with the Content of Strings ................................. 26
6.7 Mapping over Strings ............................................... 27
6.8 String Conditionals ............................................... 27
6.9 String Case Functions .............................................. 28

7 Integers (Int) .................................................. 30
7.1 Constant and Scratch Integers ..................................... 30
7.2 Integer Expressions ............................................... 30
7.3 Creating and Using Integers ...................................... 32
7.4 Viewing Integers ................................................ 32
7.5 Setting Integer Variables ......................................... 33
7.6 Integer Step Functions ............................................ 34
7.7 Integer Conditionals .............................................. 34
7.8 Integer Case Functions ............................................ 35

8 Floating Point Numbers (Fp) .................................... 37
8.1 Constant and Scratch Floating Points ............................ 37
8.2 Floating Point Expressions ....................................... 38
8.3 Creating and Using Floating Points ............................... 39
8.4 Viewing Floating Points .......................................... 39
8.5 Setting Floating Point Variables ................................ 40
8.6 Floating Point Step Functions ................................... 40
8.7 Float Point Conditionals ......................................... 41

9 Dimensions (Dim) ................................................ 42
9.1 Constant and Scratch Dimensions ................................ 42
9.2 Dimension Expressions .......................................... 42
9.3 Creating and Using Dimensions .................................. 43
9.4 Viewing Dimensions .............................................. 44
9.5 Setting Dimension Variables ..................................... 44
9.6 Dimension Step Functions ......................................... 45
9.7 Dimension Conditionals .......................................... 46
9.8 Dimension Case Functions ....................................... 47

10 Comma Separated Lists (Clist) ................................ 49
10.1 Constant and Scratch Comma Lists .............................. 49
10.2 Creating and Using Comma Lists ................................. 49
10.3 Viewing Comma Lists ............................................ 50
10.4 Setting Comma Lists ............................................. 50
10.5 Modifying Comma Lists .......................................... 51
10.6 Working with the Contents of Comma Lists .................... 52
10.7 Comma Lists as Stacks .......................................... 52
10.8 Mapping over Comma Lists ...................................... 53
10.9 Comma List Conditionals ....................................... 53
## 11 The Source Code

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1 Interfaces for Functional Programming (Prg)</td>
<td>55</td>
</tr>
<tr>
<td>11.2 Interfaces for Argument Using (Use)</td>
<td>64</td>
</tr>
<tr>
<td>11.3 Interfaces for Control Structures (Bool)</td>
<td>65</td>
</tr>
<tr>
<td>11.4 Interfaces for Token Lists (Tl)</td>
<td>66</td>
</tr>
<tr>
<td>11.5 Interfaces for Strings (Str)</td>
<td>70</td>
</tr>
<tr>
<td>11.6 Interfaces for Integers (Int)</td>
<td>73</td>
</tr>
<tr>
<td>11.7 Interfaces for Floating Point Numbers (Fp)</td>
<td>76</td>
</tr>
<tr>
<td>11.8 Interfaces for Dimensions (Dim)</td>
<td>79</td>
</tr>
<tr>
<td>11.9 Interfaces for Sorting Functions (Sort)</td>
<td>81</td>
</tr>
<tr>
<td>11.10 Interfaces for Comma Separated Lists (Clist)</td>
<td>81</td>
</tr>
</tbody>
</table>
Chapter 1

Overview of Features

1.1 Evaluation from Inside to Outside

We will compare our first example with a similar Lua example:

```latex
\PrgNewFunction \MathSquare { m } { 
  \IntSet \lTmpaInt { \IntEval {#1 * #1} } 
  \Result { \Value \lTmpaInt } 
} 
\MathSquare{5} \MathSquare{\MathSquare{5}}
```

Both examples calculate first the square of 5 and produce 25, then calculate the square of 25 and produce 625. In contrast to expl3, this functional package does evaluation of functions from inside to outside, which means composition of functions works like other programming languages such as Lua or JavaScript.

You can define new functions with \PrgNewFunction command. To make composition of functions work as expected, every function must not insert directly any token to the input stream. Instead, a function must pass the result (if any) to functional package with \Result command. And functional package is responsible for inserting result tokens to the input stream at the appropriate time.

To remove space tokens inside function code in defining functions, you’d better put function definitions inside IgnoreSpacesOn and IgnoreSpacesOff block. Within this block, ~ is used to input a space.

At the end of this section, we will compare our factorial example with a similar Lua example:

```latex
\PrgNewFunction \Fact { m } { 
  \IntCompareTF {#1} = {0} { 
    \Result {1} 
  }{ 
    \Result {\IntMathMult{#1}{\Fact{\IntMathSub{#1}{1}}}} 
  } 
} 
\Fact{4}
```

1.2 Group Scoping of Functions

In Lua language, a function or a condition expression makes a block, and the values of local variables will be reset after a block. In functional package, a condition expression is in fact a function, and you can make every function become a group by setting \Functional{scoping=true}. For example
CHAPTER 1. OVERVIEW OF FEATURES

SAME AS \texttt{expl3}, the names of local variables \textit{must} start with \texttt{l}, while names of global variables \textit{must} start with \texttt{g}. The difference is that \texttt{functional} package provides only one function for setting both local and global variables of the same type, by checking leading letters of their names. So for integer variables, you can write \texttt{\IntSet \lTmpaInt {1}} and \texttt{\IntSet \gTmpbInt {2}}.

The previous example will produce different result if we change variable from \texttt{\lTmpaInt} to \texttt{\gTmpaInt}.

As you can see, the values of global variables will never be reset after a group.

1.3 Tracing Evaluation of Functions

Since every function in \texttt{functional} package will pass its return value to the package, it is quite easy to debug your code. You can turn on the tracing by setting \texttt{\Functional{tracing=true}}. For example, the tracing log of the first example in this chapter will be the following:
1.4 Definitions of Functions

Within \texttt{expl3}, there are eight commands for defining new functions, which is good for power users.

\begin{verbatim}
\cs_new:Npn \cs_new_nopar:Npn \cs_new_protected:Npn \cs_new_protected_nopar:Npn
\end{verbatim}

Within \texttt{functional} package, there is only one command (\texttt{\PrgNewFunction}) for defining new functions, which is good for normal users. The created functions are always protected and accept \texttt{\par} in their arguments.

Since \texttt{functional} package gets the results of functions by evaluation (including expansion and execution by \TeX), it is natural to protect all functions.

1.5 Variants of Arguments

Within \texttt{expl3}, there are several expansion variants for arguments, and many expansion functions for expanding them, which are necessary for power users.

\begin{verbatim}
\module_foo:c \module_bar:e \module_bar:x \module_bar:f \module_bar:o \module_bar:V \module_bar:v
\exp_args:Nc \exp_args:Ne \exp_args:Nx \exp_args:Nf \exp_args:No \exp_args:NV \exp_args:Nv
\end{verbatim}
Within the functional package, there are only three variants (c, e, v) provided, and these variants are defined as functions (`\Name`, `\Expand`, `\Value`, respectively), which are easier to use for normal users.

```latex
\newcommand{\test}{uvw}
\Name{test}
```

```latex
\newcommand{\test}{uvw}
\Expand{111\test222}
```

```latex
\IntSet{\lTmpaInt}{123}
\Value{\lTmpaInt}
```

The most interesting feature is that you can compose these functions. For example, you can easily get the v variant of expl3 by simply composing `\Name` and `\Value` functions:

```latex
\IntSet{\lTmpaint}{123}
\Value{\Name{\lTmpaint}}
```
Chapter 2

Functional Programming (Prg)

2.1 Defining Functions and Conditionals

\texttt{\textbackslash PrgNewFunction} (\texttt{function}) \{\texttt{argument specification}\}\{\texttt{code}\}

Creates protected \texttt{(function)} for evaluating the \texttt{(code)}. Within the \texttt{(code)}, the parameters (#1, #2, etc.) will be replaced by those absorbed by the function. The returned value \texttt{must} be passed with \texttt{\Result} function. The definition is global and an error results if the \texttt{(function)} is already defined.

The \{\texttt{argument specification}\} in a list of letters, where each letter is one of the following argument specifiers (nearly all of them are \texttt{M} or \texttt{m} for functions provided by this package):

- \texttt{M} single-token argument, which will be manipulated first
- \texttt{m} multi-token argument, which will be manipulated first
- \texttt{N} single-token argument, which will not be manipulated first
- \texttt{n} multi-token argument, which will not be manipulated first

The argument manipulation for argument type \texttt{M} or \texttt{m} is: if the argument starts with a function defined with \texttt{\PrgNewFunction}, the argument will be evaluated and replaced with the returned value.

\texttt{\textbackslash PrgNewConditional} (\texttt{function}) \{\texttt{argument specification}\}\{\texttt{code}\}

Creates protected conditional \texttt{(function)} for evaluating the \texttt{(code)}. The returned value of the \texttt{(function)} \texttt{must} be either \texttt{\cTrueBool} or \texttt{\cFalseBool} and be passed with \texttt{\Result} function. The definition is global and an error results if the \texttt{(function)} is already defined.

Assume the \texttt{(function)} is \texttt{\FooIfBar}, then another function \texttt{\FooIfBarTF} will be created at the same time. \texttt{\FooIfBarTF} function has two extra arguments which are \{\texttt{true code}\} and \{\texttt{false code}\}.

2.2 Collecting Returned Values

\texttt{\Result} \{\texttt{tokens}\}

Appends \texttt{(tokens)} to \texttt{\gResultTl}, which holds the returned value of current function. This function is normally used in the \texttt{(code)} of \texttt{\PrgNewFunction} and \texttt{\PrgNewConditional}.
Chapter 3

Argument Using (Use)

3.1 Expanding Tokens

\Name {⟨control sequence name⟩}
Expands the ⟨control sequence name⟩ until only characters remain, then converts this into a control sequence and returns it. The ⟨control sequence name⟩ must consist of character tokens when exhaustively expanded.

\Value ⟨variable⟩
Recovering the content of a ⟨variable⟩ and returns the value. An error is raised if the variable does not exist or if it is invalid. Note that it is the same as \TlUse for ⟨tl var⟩, or \IntUse for ⟨int var⟩.

\Expand {⟨tokens⟩}
Expands the ⟨tokens⟩ exhaustively and returns the result.

\ExpNot {⟨tokens⟩}
Prevents expansion of the ⟨tokens⟩ inside the argument of \Expand function. The argument of \ExpNot must be surrounded by braces.

\ExpValue ⟨variable⟩
Recovering the content of the ⟨variable⟩, then prevents expansion of this material inside the argument of \Expand function.

3.2 Using Tokens

\UseOne {⟨argument⟩}
\GobbleOne {⟨argument⟩}
The function \UseOne absorbs one argument and returns it. \GobbleOne absorbs one argument and returns nothing. For example

\UseOne{abc}\GobbleOne{ijk}\UseOne{xyz} abcxyz
These functions absorb two arguments. The function \texttt{\UseGobble} discards the second argument, and returns the content of the first argument. \texttt{\GobbleUse} discards the first argument, and returns the content of the second argument. For example

\begin{verbatim}
\UseGobble{abc}{uvw}\GobbleUse{abc}{uvw}
\end{verbatim}

abcuvw
Chapter 4

Control Structures (Bool)

4.1 Constant and Scratch Booleans

\cTrueBool \cFalseBool

Constants that represent true and false, respectively. Used to implement predicates. For example

\BoolVarIfTF \cTrueBool \{\Result{True!}\} \{\Result{False!}\}
\BoolVarIfTF \cFalseBool \{\Result{True!}\} \{\Result{False!}\}

\lTmpaBool \lTmpbBool \lTmpcBool \lTmpiBool \lTmpjBool \lTmpkBool

Scratch booleans for local assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

\gTmpaBool \gTmpbBool \gTmpcBool \gTmpiBool \gTmpjBool \gTmpkBool

Scratch booleans for global assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

4.2 Creating and Setting Booleans

\BoolNew \langle boolean \rangle

Creates a new \langle boolean \rangle or raises an error if the name is already taken. The declaration is global. The \langle boolean \rangle is initially false.

\BoolConst \langle boolean \rangle \{\langle boolexpr \rangle\}

Creates a new constant \langle boolean \rangle or raises an error if the name is already taken. The value of the \langle boolean \rangle is set globally to the result of evaluating the \langle boolexpr \rangle. For example

\BoolConst \cFooSomeBool \{\IntCompare{3} > \{2\}\}
\BoolVarLog \cFooSomeBool
\BoolSet (boolean) \{\text{expr}\}

Evaluates the \{\text{expression}\} and sets the \{\text{boolean}\} variable to the logical truth of this evaluation. For example

\begin{verbatim}
\BoolSet \lTmpaBool \{\IntCompare{3}{<2}\}
\BoolVarLog \lTmpaBool
\end{verbatim}

\BoolSetTrue (boolean)

Sets \{\text{boolean}\} logically true.

\BoolSetFalse (boolean)

Sets \{\text{boolean}\} logically false.

\BoolSetEq \{\text{boolean}_1\} \{\text{boolean}_2\}

Sets \{\text{boolean}_1\} to the current value of \{\text{boolean}_2\}. For example

\begin{verbatim}
\BoolSetTrue \lTmpaBool
\BoolSetEq \lTmpbBool \lTmpaBool
\BoolVarLog \lTmpbBool
\end{verbatim}

### 4.3 Viewing Booleans

\BoolLog \{\text{expression}\}

Writes the logical truth of the \{\text{expression}\} in the log file.

\BoolVarLog \{\text{boolean}\}

Writes the logical truth of the \{\text{boolean}\} in the log file.

\BoolShow \{\text{expression}\}

Displays the logical truth of the \{\text{expression}\} on the terminal.

\BoolVarShow \{\text{boolean}\}

Displays the logical truth of the \{\text{boolean}\} on the terminal.

### 4.4 Booleans and Conditionals

\BoolIfExist \{\text{boolean}\}

\BoolIfExistTF \{\text{boolean}\} \{\text{true code}\} \{\text{false code}\}

Tests whether the \{\text{boolean}\} is currently defined. This does not check that the \{\text{boolean}\} really is a boolean variable. For example

\begin{verbatim}
\BoolIfExist \lTmpBoolean
\BoolIfExistTF \lTmpBoolean \{\text{true code}\} \{\text{false code}\}
\end{verbatim}
CHAPTER 4. CONTROL STRUCTURES (BOOL)

\BoolIfExistTF \lTmpaBool \{\Result{Yes}} \{\Result{No}}
\BoolIfExistTF \lFooUndefinedBool \{\Result{Yes}} \{\Result{No}}

\BoolVarIf ⟨boolean⟩
\BoolVarIfTF ⟨boolean⟩ {⟨true code⟩} {⟨false code⟩}

Tests the current truth of ⟨boolean⟩, and continues evaluation based on this result. For example

\BoolVarIfTrue \lTmpaBool
\BoolVarIfTF \lTmpaBool \{\Result{True!}} \{\Result{False!}}
\BoolVarIfFalse \lTmpaBool
\BoolVarIfTF \lTmpaBool \{\Result{True!}} \{\Result{False!}}

\BoolVarNot ⟨boolean⟩
\BoolVarNotTF ⟨boolean⟩ {⟨true code⟩} {⟨false code⟩}

Evaluates ⟨true code⟩ if ⟨boolean⟩ is false, and ⟨false code⟩ if ⟨boolean⟩ is true. For example

\BoolVarNotTF \IntCompare{3}>{2} \{\Result{Yes}} \{\Result{No}}

\BoolVarAnd ⟨boolean₁⟩ ⟨boolean₂⟩
\BoolVarAndTF ⟨boolean₁⟩ ⟨boolean₂⟩ {⟨true code⟩} {⟨false code⟩}

Implements the “And” operation between two booleans, hence is true if both are true. The ⟨boolean₂⟩ is only evaluated if it is needed to determine the result of \BoolVarAnd. For example

\BoolVarAndTF \IntCompare{3}>{2} \IntCompare{3}>{4} \{\Result{Yes}} \{\Result{No}}

\BoolVarOr ⟨boolean₁⟩ ⟨boolean₂⟩
\BoolVarOrTF ⟨boolean₁⟩ ⟨boolean₂⟩ {⟨true code⟩} {⟨false code⟩}

Implements the “Or” operation between two booleans, hence is true if either one is true. The ⟨boolean₂⟩ is only evaluated if it is needed to determine the result of \BoolVarOr. For example

\BoolVarOrTF \IntCompare{3}>{2} \IntCompare{3}>{4} \{\Result{Yes}} \{\Result{No}}

\BoolVarXor ⟨boolean₁⟩ ⟨boolean₂⟩
\BoolVarXorTF ⟨boolean₁⟩ ⟨boolean₂⟩ {⟨true code⟩} {⟨false code⟩}

Implements an “exclusive or” operation between two booleans. For example

\BoolVarXorTF \IntCompare{3}>{2} \IntCompare{3}>{4} \{\Result{Yes}} \{\Result{No}}
Chapter 5

Token Lists (Tl)

5.1 Constant and Scratch Token Lists

\cSpaceT1
An explicit space character contained in a token list. For use where an explicit space is required.

\cEmptyT1
Constant that is always empty.

\lTmpaTl \lTmpbTl \lTmpcTl \lTmpiTl \lTmpjTl \lTmpkTl
Scratch token lists for local assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

\gTmpaTl \gTmpbTl \gTmpcTl \gTmpiTl \gTmpjTl \gTmpkTl
Scratch token lists for global assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

5.2 Creating and Using Token Lists

\TlNew \langle tl var \rangle
Creates a new \langle tl var \rangle or raises an error if the name is already taken. The declaration is global. The \langle tl var \rangle is initially empty.

\TlConst \langle tl var \rangle \{ \langle token list \rangle \}
Creates a new constant \langle tl var \rangle or raises an error if the name is already taken. The value of the \langle tl var \rangle is set globally to the \langle token list \rangle.

\TlUse \langle tl var \rangle
Recover the content of a \langle tl var \rangle and returns the value. An error is raised if the variable does not exist or if it is invalid. Note that it is possible to use a \langle tl var \rangle directly without an accessor function.
\TlToStr \{<token list>\}

Converts the \{<token list>\} to a \{string\}, returning the resulting character tokens. A \{string\} is a series of tokens with category code 12 (other) with the exception of spaces, which retain category code 10 (space).

\TlVarToStr \{<tl var>\}

Converts the content of the \{<tl var>\} to a string, returning the resulting character tokens. A \{string\} is a series of tokens with category code 12 (other) with the exception of spaces, which retain category code 10 (space).

### 5.3 Viewing Token Lists

\TlLog \{<token list>\}

Writes the \{<token list>\} in the log file. See also \TlShow which displays the result in the terminal.

\TlVarLog \{<tl var>\}

Writes the content of the \{<tl var>\} in the log file. See also \TlVarShow which displays the result in the terminal.

\TlShow \{<token list>\}

Displays the \{<token list>\} on the terminal.

\TlVarShow \{<tl var>\}

Displays the content of the \{<tl var>\} on the terminal. This is similar to the \texttt{\show} primitive \texttt{\show}, wrapped to a fixed number of characters per line.

### 5.4 Setting Token List Variables

\TlSet \{<tl var>\} \{<tokens>\}

Sets \{<tl var>\} to contain \{<tokens>\}, removing any previous content from the variable. For example

\begin{center}
\texttt{\TlSet \lTmpiTl \{\IntMathMult{4}{5}\}}
\texttt{\TlUse \lTmpiTl}
\end{center}

\TlSetEq \{<tl var>\1 \{<tl var>\2\}

Sets the content of \{<tl var>\1\} equal to that of \{<tl var>\2\}.

\TlClear \{<tl var>\}

Clears all entries from the \{<tl var>\}. For example
\TlSet \lTmpjTl {One}
\TlClear \lTmpjTl
\TlSet \lTmpjTl {Two}
\TlUse \lTmpjTl
\Two

\TlClearNew \tl var

Ensures that the \tl var exists globally by applying \TlNew if necessary, then applies \TlClear to leave the \tl var empty.

\TlConcat \tl var_1 \tl var_2 \tl var_3

Concatenates the content of \tl var_2 and \tl var_3 together and saves the result in \tl var_1. The \tl var_2 is placed at the left side of the new token list.

\TlPutLeft \tl var \{\tokens\}

Appends \tokens to the left side of the current content of \tl var. For example

\TlSet \lTmpkTl {Functional}
\TlPutLeft \lTmpkTl {Hello}
\TlUse \lTmpkTl
HelloFunctional

\TlPutRight \tl var \{\tokens\}

Appends \tokens to the right side of the current content of \tl var. For example

\TlSet \lTmpkTl {Functional}
\TlPutRight \lTmpkTl {World}
\TlUse \lTmpkTl
FunctionalWorld

5.5 Replacing Tokens

Within token lists, replacement takes place at the top level: there is no recursion into brace groups (more precisely, within a group defined by a category code 1/2 pair).

\TlReplaceOnce \tl var \{\old tokens\} \{\new tokens\}

Replaces the first (leftmost) occurrence of \old tokens in the \tl var with \new tokens. \Old tokens cannot contain \{, \} or # (more precisely, explicit character tokens with category code 1 (begin-group) or 2 (end-group), and tokens with category code 6).

\TlReplaceAll \tl var \{\old tokens\} \{\new tokens\}

Replaces all occurrences of \old tokens in the \tl var with \new tokens. \Old tokens cannot contain \{, \} or # (more precisely, explicit character tokens with category code 1 (begin-group) or 2 (end-group), and tokens with category code 6). As this function operates from left to right, the pattern \old tokens may remain after the replacement (see \TlRemoveAll for an example).

\TlRemoveOnce \tl var \{\tokens\}

Removes the first (leftmost) occurrence of \tokens from the \tl var. \Tokens cannot contain \{, \} or # (more precisely, explicit character tokens with category code 1 (begin-group) or 2 (end-group), and
tokens with category code 6).

\template{\texttt{\textbackslash{TlRemoveAll} \langle tl var \rangle \{\langle tokens \rangle\}}}

Removes all occurrences of \langle tokens \rangle from the \langle tl var \rangle. \langle Tokens \rangle cannot contain \{, \} or \# (more precisely, explicit character tokens with category code 1 (begin-group) or 2 (end-group), and tokens with category code 6). As this function operates from left to right, the pattern \langle tokens \rangle may remain after the removal, for instance,

\begin{verbatim}
\TlSet \lTmpaTl {abcccd}
\TlRemoveAll \lTmpaTl \{bc\}
\TlUse \lTmpaTl
\end{verbatim}

\template{\texttt{\textbackslash{TlTrimSpaces} \{\langle token list \rangle\}}}

Removes any leading and trailing explicit space characters (explicit tokens with character code 32 and category code 10) from the \langle token list \rangle and returns the result.

\template{\texttt{\textbackslash{TlVarTrimSpaces} \langle tl var \rangle}}

Sets the \langle tl var \rangle to contain the result of removing any leading and trailing explicit space characters (explicit tokens with character code 32 and category code 10) from its contents.

### 5.6 Working with the Content of Token Lists

\template{\texttt{\textbackslash{TlCount} \{\langle tokens \rangle\}}}

Counts the number of \langle items \rangle in \langle tokens \rangle and returns this information. Unbraced tokens count as one element as do each token group (\{ \cdots \}). This process ignores any unprotected spaces within \langle tokens \rangle.

\template{\texttt{\textbackslash{TlVarCount} \langle tl var \rangle}}

Counts the number of \langle items \rangle in the \langle tl var \rangle and returns this information. Unbraced tokens count as one element as do each token group (\{ \cdots \}). This process ignores any unprotected spaces within the \langle tl var \rangle.

\template{\texttt{\textbackslash{TlHead} \{\langle token list \rangle\}}}

Returns the first \langle item \rangle in the \langle token list \rangle, discarding the rest of the \langle token list \rangle. All leading explicit space characters (explicit tokens with character code 32 and category code 10) are discarded; for example

\begin{verbatim}
\fbox {1\TlHead{ abc }2}
\fbox {1\TlHead{ ab }2}
\end{verbatim}

If the “head” is a brace group, rather than a single token, the braces are removed, and so

\begin{verbatim}
\fbox {\{ ab \} c }
\end{verbatim}

yields \texttt{ab}. A blank \langle token list \rangle (see \texttt{\textbackslash{TlIfBlank}}) results in \texttt{\TlHead} returning nothing.

\template{\texttt{\textbackslash{TlVarHead} \langle tl var \rangle}}

Returns the first \langle item \rangle in the \langle tl var \rangle, discarding the rest of the \langle tl var \rangle. All leading explicit space characters (explicit tokens with character code 32 and category code 10) are discarded.
\TlTail \{\langle \text{token list} \rangle\}

Discards all leading explicit space characters (explicit tokens with character code 32 and category code 10) and the first \langle \text{item} \rangle in the \langle \text{token list} \rangle, and returns the remaining tokens. Thus for example

\begin{verbatim}
\TlTail \{ a \{bc\} d \}
\end{verbatim}

and

\begin{verbatim}
\TlTail \{ a {bc} d \}
\end{verbatim}

both return \langle bc \rangle_d. A blank \langle \text{token list} \rangle (see \TlIfBlank) results in \TlTail returning nothing.

\TlVarTail \langle \text{tl var} \rangle

Discards all leading explicit space characters (explicit tokens with character code 32 and category code 10) and the first \langle \text{item} \rangle in the \langle \text{tl var} \rangle, and returns the remaining tokens.

\TlItem \{\langle \text{token list} \rangle\} \{\langle \text{integer expression} \rangle\}
\TlVarItem \langle \text{tl var} \rangle \{\langle \text{integer expression} \rangle\}

Indexing items in the \langle \text{token list} \rangle from 1 on the left, this function evaluates the \langle \text{integer expression} \rangle and returns the appropriate item from the \langle \text{token list} \rangle. If the \langle \text{integer expression} \rangle is negative, indexing occurs from the right of the token list, starting at $-1$ for the right-most item. If the index is out of bounds, then the function returns nothing.

\TlRandItem \{\langle \text{token list} \rangle\}
\TlVarRandItem \langle \text{tl var} \rangle

Selects and returns a pseudo-random item of the \langle \text{token list} \rangle. If the \langle \text{token list} \rangle is blank, the result is empty.

### 5.7 Mapping over Token Lists

All mappings are done at the current group level, i.e., any local assignments made by the \langle \text{function} \rangle or \langle \text{code} \rangle discussed below remain in effect after the loop.

\TlMapInline \{\langle \text{token list} \rangle\} \{\langle \text{inline function} \rangle\}

Applies the \langle \text{inline function} \rangle to every \langle \text{item} \rangle stored within the \langle \text{token list} \rangle. The \langle \text{inline function} \rangle should consist of code which receives the \langle \text{item} \rangle as \#1.

\TlVarMapInline \langle \text{tl var} \rangle \{\langle \text{inline function} \rangle\}

Applies the \langle \text{inline function} \rangle to every \langle \text{item} \rangle stored within the \langle \text{tl var} \rangle. The \langle \text{inline function} \rangle should consist of code which receives the \langle \text{item} \rangle as \#1.

\TlMapVariable \{\langle \text{token list} \rangle\} \langle \text{variable} \rangle \{\langle \text{code} \rangle\}

Stores each \langle \text{item} \rangle of the \langle \text{token list} \rangle in turn in the \langle \text{token list} \rangle \langle \text{variable} \rangle and applies the \langle \text{code} \rangle. The \langle \text{code} \rangle will usually make use of the \langle \text{variable} \rangle, but this is not enforced. The assignments to the \langle \text{variable} \rangle are local. Its value after the loop is the last \langle \text{item} \rangle in the \langle \text{tl var} \rangle, or its original value if the \langle \text{tl var} \rangle is blank.
\texttt{\textbackslash TlVarMapVariable} \langle \textbackslash tl \ var \rangle \langle \textbackslash variable \rangle \{\langle \textcode \rangle \}

Stores each \langle \textitem \rangle of the \langle \textbackslash tl \ var \rangle in turn in the \langle \textbackslash variable \rangle and applies the \langle \textcode \rangle. The \langle \textcode \rangle will usually make use of the \langle \textbackslash variable \rangle, but this is not enforced. The assignments to the \langle \textbackslash variable \rangle are local. Its value after the loop is the last \langle \textitem \rangle in the \langle \textbackslash tl \ var \rangle, or its original value if the \langle \textbackslash tl \ var \rangle is blank.

\section{Token List Conditionals}

\texttt{\textbackslash TlIfExist} \langle \textbackslash tl \ var \rangle
\texttt{\textbackslash TlIfExistTF} \langle \textbackslash tl \ var \rangle \{\langle \texttrue \ \textcode \rangle \} \{\langle \textfalse \ \textcode \rangle \}

Tests whether the \langle \textbackslash tl \ var \rangle is currently defined. This does not check that the \langle \textbackslash tl \ var \rangle really is a token list variable.

\texttt{\textbackslash TlIfEmpty} \{\langle \texttoken \ \textlist \rangle \}
\texttt{\textbackslash TlIfEmptyTF} \{\langle \texttoken \ \textlist \rangle \} \{\langle \texttrue \ \textcode \rangle \} \{\langle \textfalse \ \textcode \rangle \}

Tests if the \langle \texttoken \ \textlist \rangle is entirely empty (i.e. contains no tokens at all). For example

\begin{verbatim}
\texttt{\textbackslash TlIfEmptyTF} \{abc\} \{\texttt{\textbackslash Result\{Empty\}}\} \{\texttt{\textbackslash Result\{NonEmpty\}}\}
\texttt{\textbackslash TlIfEmptyTF} {} \{\texttt{\textbackslash Result\{Empty\}}\} \{\texttt{\textbackslash Result\{NonEmpty\}}\}
\end{verbatim}

\texttt{NonEmpty Empty}

\texttt{\textbackslash TlVarIfEmpty} \langle \textbackslash tl \ var \rangle
\texttt{\textbackslash TlVarIfEmptyTF} \langle \textbackslash tl \ var \rangle \{\langle \texttrue \ \textcode \rangle \} \{\langle \textfalse \ \textcode \rangle \}

Tests if the \langle \texttoken \ \textlist \ \textvariable \rangle is entirely empty (i.e. contains no tokens at all). For example

\begin{verbatim}
\texttt{\textbackslash TlSet} \\texttt{\lTmpaTl} \{abc\}
\texttt{\textbackslash TlVarIfEmptyTF} \\texttt{\lTmpaTl} \{\texttt{\textbackslash Result\{Empty\}}\} \{\texttt{\textbackslash Result\{NonEmpty\}}\}
\texttt{\textbackslash TlClear} \\texttt{\lTmpaTl}
\texttt{\textbackslash TlVarIfEmptyTF} \\texttt{\lTmpaTl} \{\texttt{\textbackslash Result\{Empty\}}\} \{\texttt{\textbackslash Result\{NonEmpty\}}\}
\end{verbatim}

\texttt{NonEmpty Empty}

\texttt{\textbackslash TlIfBlank} \{\langle \texttoken \ \textlist \rangle \}
\texttt{\textbackslash TlIfBlankTF} \{\langle \texttoken \ \textlist \rangle \} \{\langle \texttrue \ \textcode \rangle \} \{\langle \textfalse \ \textcode \rangle \}

Tests if the \langle \texttoken \ \textlist \rangle consists only of blank spaces (i.e. contains no item). The test is \texttrue if \langle \texttoken \ \textlist \rangle is zero or more explicit space characters (explicit tokens with character code 32 and category code 10), and is \textfalse otherwise.

\texttt{\textbackslash TlIfEq} \{\langle \texttoken \ \textlist \rangle \} \{\langle \texttoken \ \textlist \rangle \}
\texttt{\textbackslash TlIfEqTF} \{\langle \texttoken \ \textlist \rangle \} \{\langle \texttoken \ \textlist \rangle \} \{\langle \texttrue \ \textcode \rangle \} \{\langle \textfalse \ \textcode \rangle \}

Tests if \langle \texttoken \ \textlist \rangle and \langle \texttoken \ \textlist \rangle contain the same list of tokens, both in respect of character codes and category codes. See \texttt{\textbackslash StrIfEq} if category codes are not important. For example

\begin{verbatim}
\texttt{\textbackslash TlIfEqTF} \{abc\} \{abc\} \{\texttt{\textbackslash Result\{Yes\}}\} \{\texttt{\textbackslash Result\{No\}}\}
\texttt{\textbackslash TlIfEqTF} \{abc\} \{xyz\} \{\texttt{\textbackslash Result\{Yes\}}\} \{\texttt{\textbackslash Result\{No\}}\}
\end{verbatim}

\texttt{Yes No}

\texttt{\textbackslash TlVarIfEq} \langle \textbackslash tl \ var \rangle \langle \textbackslash tl \ var \rangle
\texttt{\textbackslash TlVarIfEqTF} \langle \textbackslash tl \ var \rangle \langle \textbackslash tl \ var \rangle \{\langle \texttrue \ \textcode \rangle \} \{\langle \textfalse \ \textcode \rangle \}

Compares the content of two \langle \texttoken \ \textlist \ \textvariables \rangle and is logically \texttrue if the two contain the same
list of tokens (i.e., identical in both the list of characters they contain and the category codes of those characters). For example

\TlSet {abc}
\TlSet {abc}
\TlSet {xyz}
\TlVarIfEqTF \lTmpaTl \lTmpbTl \Result{Yes} \Result{No}
\TlVarIfEqTF \lTmpaTl \lTmpcTl \Result{Yes} \Result{No}

See also \StrVarIfEq for a comparison that ignores category codes.

\TlIfIn {⟨token list₁⟩} {⟨token list₂⟩}
\TlIfInTF {⟨token list₁⟩} {⟨token list₂⟩} {⟨true code⟩} {⟨false code⟩}

Tests if ⟨token list₂⟩ is found inside ⟨token list₁⟩. The ⟨token list₂⟩ cannot contain the tokens {, } or # (more precisely, explicit character tokens with category code 1 (begin-group) or 2 (end-group), and tokens with category code 6). The search does not enter brace (category code 1/2) groups.

\TlVarIfIn {⟨tl var⟩} {⟨token list⟩}
\TlVarIfInTF {⟨tl var⟩} {⟨token list⟩} {⟨true code⟩} {⟨false code⟩}

Tests if the ⟨token list⟩ is found in the content of the ⟨tl var⟩. The ⟨token list⟩ cannot contain the tokens {, } or # (more precisely, explicit character tokens with category code 1 (begin-group) or 2 (end-group), and tokens with category code 6).

\TlIfSingle {⟨token list⟩}
\TlIfSingleTF {⟨token list⟩} {⟨true code⟩} {⟨false code⟩}

Tests if the ⟨token list⟩ has exactly one ⟨item⟩, i.e., is a single normal token (neither an explicit space character nor a begin-group character) or a single brace group, surrounded by optional spaces on both sides. In other words, such a token list has token count 1 according to \TlCount.

\TlVarIfSingle {⟨tl var⟩}
\TlVarIfSingleTF {⟨tl var⟩} {⟨true code⟩} {⟨false code⟩}

Tests if the content of the ⟨tl var⟩ consists of a single ⟨item⟩, i.e., is a single normal token (neither an explicit space character nor a begin-group character) or a single brace group, surrounded by optional spaces on both sides. In other words, such a token list has token count 1 according to \TlVarCount.

### 5.9 Token List Case Functions

\TlVarCase {⟨test token list variable⟩}
{  ⟨token list variable case₁⟩ {⟨code case₁⟩}
  ⟨token list variable case₂⟩ {⟨code case₂⟩}
  ...  ⟨token list variable caseₙ⟩ {⟨code caseₙ⟩}
}

This function compares the ⟨test token list variable⟩ in turn with each of the ⟨token list variable cases⟩. If the two are equal (as described for \TlVarIfEq) then the associated ⟨code⟩ is left in the input stream and other cases are discarded. The function does nothing if there is no match.
This function compares the \( \text{<test token list variable>} \) in turn with each of the \( \text{(token list variable cases)} \). If the two are equal (as described for \( \text{TlVarIfEq} \)) then the associated \( \text{(code)} \) is left in the input stream and other cases are discarded. If any of the cases are matched, the \( \text{<true code>} \) is also inserted into the input stream (after the code for the appropriate case).

This function compares the \( \text{<test token list variable>} \) in turn with each of the \( \text{(token list variable cases)} \). If none match then the \( \text{<false code>} \) is inserted into the input stream (after the code for the appropriate case).

This function compares the \( \text{<test token list variable>} \) in turn with each of the \( \text{(token list variable cases)} \). If the two are equal (as described for \( \text{TlVarIfEq} \)) then the associated \( \text{(code)} \) is left in the input stream and other cases are discarded. If any of the cases are matched, the \( \text{<true code>} \) is also inserted into the input stream (after the code for the appropriate case), while if none match then the \( \text{<false code>} \) is inserted. The function \( \text{TlVarCase} \), which does nothing if there is no match, is also available.
Chapter 6

Strings (Str)

6.1 Constant and Scratch Strings

\cAmpersandStr \cAtsignStr \cBackslashStr \cLeftBraceStr \cRightBraceStr \cCircumflexStr \cColonStr \cDollarStr \cHashStr \cPercentStr \cTildeStr \cUnderscoreStr \cZeroStr

Constant strings, containing a single character token, with category code 12.

\lTmpaStr \lTmpbStr \lTmpcStr \lTmpiStr \lTmpjStr \lTmpkStr

Scratch strings for local assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

\gTmpaStr \gTmpbStr \gTmpcStr \gTmpiStr \gTmpjStr \gTmpkStr

Scratch strings for global assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

6.2 Creating and Using Strings

\StrNew (str var)

Creates a new (str var) or raises an error if the name is already taken. The declaration is global. The (str var) is initially empty.

\StrConst (str var) {({token list})}

Creates a new constant (str var) or raises an error if the name is already taken. The value of the (str var) is set globally to the (token list), converted to a string.

\StrUse (str var)

Recover the content of a (str var) and returns the value. An error is raised if the variable does not exist or if it is invalid. Note that it is possible to use a (str) directly without an accessor function.
6.3 Viewing Strings

\StrLog {⟨token list⟩}

Writes ⟨token list⟩ in the log file.

\StrVarLog ⟨str var⟩

Writes the content of the ⟨str var⟩ in the log file. For example

\StrSet \lTmpiStr {1234\abcd5678}
\StrVarLog \lTmpiStr

\StrShow {⟨token list⟩}

Displays ⟨token list⟩ on the terminal.

\StrVarShow ⟨str var⟩

Displays the content of the ⟨str var⟩ on the terminal.

6.4 Setting String Variables

\StrSet ⟨str var⟩ {⟨token list⟩}

Converts the ⟨token list⟩ to a ⟨string⟩, and stores the result in ⟨str var⟩. For example

\StrSet \lTmpiStr {\IntMathMult{4}{5}}
\StrUse \lTmpiStr

\StrSetEq ⟨str var₁⟩ ⟨str var₂⟩

Sets the content of ⟨str var₁⟩ equal to that of ⟨str var₂⟩.

\StrClear ⟨str var⟩

Clears the content of the ⟨str var⟩. For example

\StrSet \lTmpjStr {One}
\StrClear \lTmpjStr
\StrSet \lTmpjStr {Two}
\StrUse \lTmpjStr

\StrClearNew ⟨str var⟩

Ensures that the ⟨str var⟩ exists globally by applying \StrNew if necessary, then applies \StrClear to leave the ⟨str var⟩ empty.
\StrConcat \langle str\ var_1 \rangle \langle str\ var_2 \rangle \langle str\ var_3 \rangle

Concatenates the content of \langle str\ var_2 \rangle and \langle str\ var_3 \rangle together and saves the result in \langle str\ var_1 \rangle. The \langle str\ var_2 \rangle is placed at the left side of the new string variable. The \langle str\ var_2 \rangle and \langle str\ var_3 \rangle must indeed be strings, as this function does not convert their contents to a string.

\StrPutLeft \langle str\ var \rangle \{\langle token\ list \rangle\}

Converts the \langle token\ list \rangle to a \langle string \rangle, and prepends the result to \langle str\ var \rangle. The current contents of the \langle str\ var \rangle are not automatically converted to a string. For example

\StrSet \lTmpkStr \{Functional\}
\StrPutLeft \lTmpkStr \{Hello\}
\StrUse \lTmpkStr

HelloFunctional

\StrPutRight \langle str\ var \rangle \{\langle token\ list \rangle\}

Converts the \langle token\ list \rangle to a \langle string \rangle, and appends the result to \langle str\ var \rangle. The current contents of the \langle str\ var \rangle are not automatically converted to a string. For example

\StrSet \lTmpkStr \{Functional\}
\StrPutRight \lTmpkStr \{World\}
\StrUse \lTmpkStr

FunctionalWorld

6.5 Modifying String Variables

\StrReplaceOnce \langle str\ var \rangle \{\langle old \rangle\} \{\langle new \rangle\}

Converts the \langle old \rangle and \langle new \rangle token lists to strings, then replaces the first (leftmost) occurrence of \langle old string \rangle in the \langle str\ var \rangle with \langle new string \rangle.

\StrReplaceAll \langle str\ var \rangle \{\langle old \rangle\} \{\langle new \rangle\}

Converts the \langle old \rangle and \langle new \rangle token lists to strings, then replaces all occurrences of \langle old string \rangle in the \langle str\ var \rangle with \langle new string \rangle. As this function operates from left to right, the pattern \langle old string \rangle may remain after the replacement.

\StrRemoveOnce \langle str\ var \rangle \{\langle token\ list \rangle\}

Converts the \langle token\ list \rangle to a \langle string \rangle then removes the first (leftmost) occurrence of \langle string \rangle from the \langle str\ var \rangle.

\StrRemoveAll \langle str\ var \rangle \{\langle token\ list \rangle\}

Converts the \langle token\ list \rangle to a \langle string \rangle then removes all occurrences of \langle string \rangle from the \langle str\ var \rangle. As this function operates from left to right, the pattern \langle string \rangle may remain after the removal, for instance,
6.6 Working with the Content of Strings

\StrCount \{\langle token list\rangle\}

Returns the number of characters in the string representation of \langle token list\rangle, as an integer denotation. All characters including spaces are counted. Due to naming conflict, you need to use \StrSize instead of \StrCount if you want to use functional package together with xstring package.

\StrVarCount \langle tl var\rangle

Returns the number of characters in the string representation of the \langle tl var\rangle, as an integer denotation. All characters including spaces are counted.

\StrHead \{\langle token list\rangle\}

Converts the \langle token list\rangle into a \langle string\rangle. The first character in the \langle string\rangle is then returned, with category code “other”. If the first character is a space, it returns a space token with category code 10 (blank space). If the \langle string\rangle is empty, then nothing is returned.

\StrVarHead \langle tl var\rangle

Converts the \langle tl var\rangle into a \langle string\rangle. The first character in the \langle string\rangle is then returned, with category code “other”. If the first character is a space, it returns a space token with category code 10 (blank space). If the \langle string\rangle is empty, then nothing is returned.

\StrTail \{\langle token list\rangle\}

Converts the \langle token list\rangle to a \langle string\rangle, removes the first character, and returns the remaining characters (if any) with category codes 12 and 10 (for spaces). If the first character is a space, it only trims that space. If the \langle token list\rangle is empty, then nothing is left on the input stream.

\StrVarTail \langle tl var\rangle

Converts the \langle tl var\rangle to a \langle string\rangle, removes the first character, and returns the remaining characters (if any) with category codes 12 and 10 (for spaces). If the first character is a space, it only trims that space. If the \langle token list\rangle is empty, then nothing is left on the input stream.

\StrItem \{\langle token list\rangle\} \{\langle integer expression\rangle\}

Converts the \langle token list\rangle to a \langle string\rangle, and returns the character in position \langle integer expression\rangle of the \langle string\rangle, starting at 1 for the first (left-most) character. All characters including spaces are taken into account. If the \langle integer expression\rangle is negative, characters are counted from the end of the \langle string\rangle. Hence, −1 is the right-most character, etc.

\StrVarItem \langle tl var\rangle \{\langle integer expression\rangle\}

Converts the \langle tl var\rangle to a \langle string\rangle, and returns the character in position \langle integer expression\rangle of the \langle string\rangle, starting at 1 for the first (left-most) character. All characters including spaces are taken into account. If the \langle integer expression\rangle is negative, characters are counted from the end of the \langle string\rangle. Hence, −1 is the right-most character, etc.
CHAPTER 6. STRINGS (str)

6.7 Mapping over Strings

All mappings are done at the current group level, i.e. any local assignments made by the (function) or (code) discussed below remain in effect after the loop.

\StrMapInline \token list \inline function
\StrVarMapInline \str var \inline function

Converts the \token list to a \string then applies the \inline function to every \character in the \str var including spaces. The \inline function should consist of code which receives the \character as #1.

\StrMapVariable \token list \variable \code
\StrVarMapVariable \str var \variable \code

Converts the \token list to a \string then stores each \character in the \string (including spaces) in turn in the \string or token list \variable and applies the \code. The \code will usually make use of the \variable, but this is not enforced. The assignments to the \variable are local. Its value after the loop is the last \character in the \string, or its original value if the \string is empty.

6.8 String Conditionals

\StrIfExist \str var
\StrIfExistTF \str var \true code \false code

Tests whether the \str var is currently defined. This does not check that the \str var really is a string.

\StrVarIfEmpty \str var
\StrVarIfEmptyTF \str var \true code \false code

Tests if the \string variable is entirely empty (i.e. contains no characters at all). For example

\StrSet \lTmpaStr {abc}
\StrVarIfEmptyTF \lTmpaStr {Result {Empty}} {Result {NonEmpty}}
\StrClear \lTmpaStr
\StrVarIfEmptyTF \lTmpaStr {Result {Empty}} {Result {NonEmpty}}

\StrIfEq \tl1 \tl2
\StrIfEqTF \tl1 \tl2 \true code \false code

Compares the two \token lists on a character by character basis (namely after converting them to strings), and is \true if the two \strings contain the same characters in the same order. See \TlIfEq to compare tokens (including their category codes) rather than characters. For example

\StrIfEqTF {abc} {abc} {Result {Yes}} {Result {No}}
\StrIfEqTF {abc} {xyz} {Result {Yes}} {Result {No}}

\StrVarIfEq \str var1 \str var2
\StrVarIfEqTF \str var1 \str var2 \true code \false code

Compares the content of two \str variables and is logically \true if the two contain the same characters in the same order. See \TlVarIfEq to compare tokens (including their category codes) rather than characters.
\StrSet \lTmpaStr {abc}
\StrSet \lTmpbStr {abc}
\StrSet \lTmpcStr {xyz}
\StrVarIfEqTF \lTmpaStr \lTmpbStr {\Result{Yes}} {\Result{No}}
\StrVarIfEqTF \lTmpaStr \lTmpcStr {\Result{Yes}} {\Result{No}}
\StrIfInTF ⟨tl₁⟩ {⟨tl₂⟩} {⟨true code⟩} {⟨false code⟩}
\StrVarIfInTF ⟨str var⟩ {⟨token list⟩} {⟨true code⟩} {⟨false code⟩}
\StrCompare ⟨⟨tl₁⟩⟩ ⟨⟨relation⟩⟩ ⟨⟨tl₂⟩⟩
\StrCompareTF ⟨⟨tl₁⟩⟩ ⟨⟨relation⟩⟩ ⟨⟨tl₂⟩⟩ {⟨true code⟩} {⟨false code⟩}
\StrCase ⟨⟨test string⟩⟩
{ ⟨⟨string case₁⟩⟩ ⟨⟨code case₁⟩⟩
  ⟨⟨string case₂⟩⟩ ⟨⟨code case₂⟩⟩
  ... 
  ⟨⟨string caseₙ⟩⟩ ⟨⟨code caseₙ⟩⟩
}

Compares both ⟨token lists⟩ to ⟨strings⟩ and tests whether ⟨string₂⟩ is found inside ⟨string₁⟩.

\StrVarIfInTF ⟨str var⟩ {⟨token list⟩}
\StrVarIfInTF ⟨str var⟩ {⟨token list⟩} {⟨true code⟩} {⟨false code⟩}

Converts the ⟨token list⟩ to a ⟨string⟩ and tests if that ⟨string⟩ is found in the content of the ⟨str var⟩.

\StrCompare ⟨⟨tl₁⟩⟩ ⟨⟨relation⟩⟩ ⟨⟨tl₂⟩⟩
\StrCompareTF ⟨⟨tl₁⟩⟩ ⟨⟨relation⟩⟩ ⟨⟨tl₂⟩⟩ {⟨true code⟩} {⟨false code⟩}

Compares the two ⟨token lists⟩ on a character by character basis (namely after converting them to strings) in a lexicographic order according to the character codes of the characters. The ⟨relation⟩ can be <, =, or > and the test is true under the following conditions:

- for <, if the first string is earlier than the second in lexicographic order;
- for =, if the two strings have exactly the same characters;
- for >, if the first string is later than the second in lexicographic order.

For example:
\StrCompareTF {ab} < {abc} {\Result{Yes}} {\Result{No}}
\StrCompareTF {ab} < {aa} {\Result{Yes}} {\Result{No}}

Due to naming conflict, you need to use \StrIfCompare/\StrIfCompareTF as a replacement if you want to use functional package together with xstring package.

### 6.9 String Case Functions

\StrCase ⟨⟨test string⟩⟩
{ ⟨langle string case₁⟩⟩ ⟨langle code case₁⟩⟩
  ⟨langle string case₂⟩⟩ ⟨langle code case₂⟩⟩
  ... 
  ⟨langle string caseₙ⟩⟩ ⟨langle code caseₙ⟩⟩
}

Compares the ⟨test string⟩ in turn with each of the ⟨string cases⟩ (all token lists are converted to strings). If the two are equal (as described for \StrIfEq) then the associated ⟨code⟩ is left in the input stream and other cases are discarded.
\StrCaseT \{(test string)\}
  \{
    \{(string case_1)\} \{(code case_1)\}
    \{(string case_2)\} \{(code case_2)\}
    \ldots
    \{(string case_n)\} \{(code case_n)\}
  \}
  \{(true code)\}

Comparates the \textit{(test string)} in turn with each of the \textit{(string cases)} (all token lists are converted to strings). If the two are equal (as described for \StrIfEq) then the associated \textit{(code)} is left in the input stream and other cases are discarded. If any of the cases are matched, the \textit{(true code)} is also inserted into the input stream (after the code for the appropriate case).

\StrCaseF \{(test string)\}
  \{
    \{(string case_1)\} \{(code case_1)\}
    \{(string case_2)\} \{(code case_2)\}
    \ldots
    \{(string case_n)\} \{(code case_n)\}
  \}
  \{(false code)\}

Comparates the \textit{(test string)} in turn with each of the \textit{(string cases)} (all token lists are converted to strings). If the two are equal (as described for \StrIfEq) then the associated \textit{(code)} is left in the input stream and other cases are discarded. If none match then the \textit{(false code)} is inserted.

\StrCaseTF \{(test string)\}
  \{
    \{(string case_1)\} \{(code case_1)\}
    \{(string case_2)\} \{(code case_2)\}
    \ldots
    \{(string case_n)\} \{(code case_n)\}
  \}
  \{(true code)\}
  \{(false code)\}

Comparates the \textit{(test string)} in turn with each of the \textit{(string cases)} (all token lists are converted to strings). If the two are equal (as described for \StrIfEq) then the associated \textit{(code)} is left in the input stream and other cases are discarded. If any of the cases are matched, the \textit{(true code)} is also inserted into the input stream (after the code for the appropriate case), while if none match then the \textit{(false code)} is inserted.
Chapter 7

Integers (Int)

7.1 Constant and Scratch Integers

\cZeroInt \cOneInt

Integer values used with primitive tests and assignments: their self-terminating nature makes these more convenient and faster than literal numbers.

\cMaxInt

The maximum value that can be stored as an integer.

\cMaxRegisterInt

Maximum number of registers.

\cMaxCharInt

Maximum character code completely supported by the engine.

\lTmpaInt \lTmpbInt \lTmpcInt \lTmpiInt \lTmpjInt \lTmpkInt

Scratch integer for local assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

\gTmpaInt \gTmpbInt \gTmpcInt \gTmpiInt \gTmpjInt \gTmpkInt

Scratch integer for global assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

7.2 Integer Expressions

\IntEval \{integer expression\}

Evaluates the \{integer expression\} and returns the result: for positive results an explicit sequence of decimal digits not starting with 0, for negative results – followed by such a sequence, and 0 for zero. For example
\IntEval{(1+4)\ast(2-3)/5} \quad -1

\IntMathAdd{\langle integer\ expression_1\rangle}{\langle integer\ expression_2\rangle}

Adds \langle integer\ expression_1\rangle and \langle integer\ expression_2\rangle, and returns the result. For example
\IntMathAdd{\langle 7\rangle}{\langle 3\rangle} \quad 10

\IntMathSub{\langle integer\ expression_1\rangle}{\langle integer\ expression_2\rangle}

Subtracts \langle integer\ expression_2\rangle from \langle integer\ expression_1\rangle, and returns the result. For example
\IntMathSub{\langle 7\rangle}{\langle 3\rangle} \quad 4

\IntMathMult{\langle integer\ expression_1\rangle}{\langle integer\ expression_2\rangle}

Multiplies \langle integer\ expression_1\rangle by \langle integer\ expression_2\rangle, and returns the result. For example
\IntMathMult{\langle 7\rangle}{\langle 3\rangle} \quad 21

\IntMathDiv{\langle integer\ expression_1\rangle}{\langle integer\ expression_2\rangle}

Evaluates the two \langle integer\ expressions\rangle as described earlier, then divides the first value by the second, and rounds the result to the closest integer. Ties are rounded away from zero. Note that this is identical to using / directly in an \langle integer\ expression\rangle. The result is returned as an \langle integer\ denotation\rangle. For example
\IntMathDiv{\langle 8\rangle}{\langle 3\rangle} \quad 3

\IntMathDivTruncate{\langle integer\ expression_1\rangle}{\langle integer\ expression_2\rangle}

Evaluates the two \langle integer\ expressions\rangle as described earlier, then divides the first value by the second, and rounds the result towards zero. Note that division using / rounds to the closest integer instead. The result is returned as an \langle integer\ denotation\rangle. For example
\IntMathDivTruncate{\langle 8\rangle}{\langle 3\rangle} \quad 2

\IntMathSign{\langle integer\ expression\rangle}

Evaluates the \langle integer\ expression\rangle then leaves 1 or 0 or \(-1\) in the input stream according to the sign of the result.

\IntMathAbs{\langle integer\ expression\rangle}

Evaluates the \langle integer\ expression\rangle as described for \IntEval and leaves the absolute value of the result in the input stream as an \langle integer\ denotation\rangle after two expansions.

\IntMathMax{\langle integer\ expression_1\rangle}{\langle integer\ expression_2\rangle}
\IntMathMin{\langle integer\ expression_1\rangle}{\langle integer\ expression_2\rangle}

Evaluates the \langle integer\ expressions\rangle as described for \IntEval and leaves either the larger or smaller value in the input stream as an \langle integer\ denotation\rangle after two expansions.
\texttt{\texttt{\texttt{\IntMathMod\{(intexpr_1\}}\texttt{\texttt{\}} \texttt{\texttt{\{intexpr_2\}}\}}} \\
Evaluates the two \langle integer expressions \rangle as described earlier, then calculates the integer remainder of dividing the first expression by the second. This is obtained by subtracting \texttt{\IntMathDivTruncate\{(intexpr_1\}} \texttt{\texttt{\}} \texttt{\texttt{\{intexpr_2\}}\} times \langle intexpr_2 \rangle from \langle intexpr_1 \rangle. Thus, the result has the same sign as \langle intexpr_1 \rangle and its absolute value is strictly less than that of \langle intexpr_2 \rangle. The result is left in the input stream as an \langle integer denotation \rangle after two expansions.

\texttt{\texttt{\texttt{\IntMathRand\{(intexpr_1\}}\texttt{\texttt{\}} \texttt{\texttt{\{intexpr_2\}}\}}} \\
Evaluates the two \langle integer expressions \rangle and produces a pseudo-random number between the two (with bounds included).

### 7.3 Creating and Using Integers

\texttt{\texttt{\texttt{\IntNew\{integer\}}} \\
Creates a new \langle integer \rangle or raises an error if the name is already taken. The declaration is global. The \langle integer \rangle is initially equal to 0.

\texttt{\texttt{\texttt{\IntConst\{integer\} \{\langle integer expression \rangle\}}}} \\
Creates a new constant \langle integer \rangle or raises an error if the name is already taken. The value of the \langle integer \rangle is set globally to the \langle integer expression \rangle.

\texttt{\texttt{\texttt{\IntUse\{integer\}}} \\
Recovers the content of an \langle integer \rangle and returns the value. An error is raised if the variable does not exist or if it is invalid.

### 7.4 Viewing Integers

\texttt{\texttt{\texttt{\IntLog\{\langle integer expression \rangle\}}} \\
Writes the result of evaluating the \langle integer expression \rangle in the log file.

\texttt{\texttt{\texttt{\IntVarLog\{integer\}}} \\
Writes the value of the \langle integer \rangle in the log file.

\texttt{\texttt{\texttt{\IntShow\{\langle integer expression \rangle\}}} \\
Displays the result of evaluating the \langle integer expression \rangle on the terminal.

\texttt{\texttt{\texttt{\IntVarShow\{integer\}}} \\
Displays the value of the \langle integer \rangle on the terminal.
7.5 Setting Integer Variables

\begin{verbatim}
\IntSet {integer} \{integer expression\}
\end{verbatim}
Sets \( \text{integer} \) to the value of \( \text{integer expression} \), which must evaluate to an integer (as described for \IntEval). For example

\begin{verbatim}
\IntSet \lTmpaInt {3+5}
\IntUse \lTmpaInt
\end{verbatim}

\begin{verbatim}
\IntSetEq \lTmpaInt {integer_1} {integer_2}
\end{verbatim}
Sets the content of \( \text{integer}_1 \) equal to that of \( \text{integer}_2 \).

\begin{verbatim}
\IntZero {integer}
\end{verbatim}
Sets \( \text{integer} \) to 0. For example

\begin{verbatim}
\IntSet \lTmpaInt {5}
\IntZero \lTmpaInt
\IntUse \lTmpaInt
\end{verbatim}

\begin{verbatim}
\IntZeroNew {integer}
\end{verbatim}
Ensures that the \( \text{integer} \) exists globally by applying \IntNew if necessary, then applies \IntZero to leave the \( \text{integer} \) set to zero.

\begin{verbatim}
\IntIncr {integer}
\end{verbatim}
Increases the value stored in \( \text{integer} \) by 1. For example

\begin{verbatim}
\IntSet \lTmpaInt {5}
\IntIncr \lTmpaInt
\IntUse \lTmpaInt
\end{verbatim}

\begin{verbatim}
\IntDecr {integer}
\end{verbatim}
Decreases the value stored in \( \text{integer} \) by 1. For example

\begin{verbatim}
\IntSet \lTmpaInt {5}
\IntDecr \lTmpaInt
\IntUse \lTmpaInt
\end{verbatim}

\begin{verbatim}
\IntAdd {integer} \{integer expression\}
\end{verbatim}
Adds the result of the \( \text{integer expression} \) to the current content of the \( \text{integer} \). For example

\begin{verbatim}
\IntSet \lTmpaInt {5}
\IntAdd \lTmpaInt {2}
\IntUse \lTmpaInt
\end{verbatim}
\IntSub \langle integer \rangle \{ \langle integer expression \rangle \}

Subtracts the result of the \langle integer expression \rangle from the current content of the \langle integer \rangle. For example

\IntSet \lTmpaInt \{5\}
\IntSub \lTmpaInt \{3\}
\IntUse \lTmpaInt

7.6 Integer Step Functions

\IntStepInline \{ \langle initial value \rangle \} \{ \langle step \rangle \} \{ \langle final value \rangle \} \{ \langle code \rangle \}

This function first evaluates the \langle initial value \rangle, \langle step \rangle and \langle final value \rangle, all of which should be integer expressions. Then for each \langle value \rangle from the \langle initial value \rangle to the \langle final value \rangle in turn (using \langle step \rangle between each \langle value \rangle), the \langle code \rangle is inserted into the input stream with \#1 replaced by the current \langle value \rangle. Thus the \langle code \rangle should define a function of one argument (\#1). For example

\IgnoreSpacesOn
\TlClear \lTmpaTl
\IntStepInline \{1\} \{3\} \{30\} {\TlPutRight \lTmpaTl {{\#1}}} 
\Result \{\Value\lTmpaTl\}
\IgnoreSpacesOff

produces \[1\][4][7][10][13][16][19][22][25][28].

\IntStepVariable \{ \langle initial value \rangle \} \{ \langle step \rangle \} \{ \langle final value \rangle \} \{ \langle tl var \rangle \} \{ \langle code \rangle \}

This function first evaluates the \langle initial value \rangle, \langle step \rangle and \langle final value \rangle, all of which should be integer expressions. Then for each \langle value \rangle from the \langle initial value \rangle to the \langle final value \rangle in turn (using \langle step \rangle between each \langle value \rangle), the \langle code \rangle is evaluated, with the \langle tl var \rangle defined as the current \langle value \rangle. Thus the \langle code \rangle should make use of the \langle tl var \rangle.

7.7 Integer Conditionals

\IntIfExist \langle integer \rangle
\IntIfExistTF \langle integer \rangle \{ \langle true code \rangle \} \{ \langle false code \rangle \}

Tests whether the \langle integer \rangle is currently defined. This does not check that the \langle integer \rangle really is an integer variable.

\IntIfOdd \{ \langle integer expression \rangle \}
\IntIfOddTF \{ \langle integer expression \rangle \} \{ \langle true code \rangle \} \{ \langle false code \rangle \}

This function first evaluates the \langle integer expression \rangle as described for \IntEval. It then evaluates if this is odd or even, as appropriate.

\IntIfEven \{ \langle integer expression \rangle \}
\IntIfEvenTF \{ \langle integer expression \rangle \} \{ \langle true code \rangle \} \{ \langle false code \rangle \}

This function first evaluates the \langle integer expression \rangle as described for \IntEval. It then evaluates if this is even or odd, as appropriate.
This function first evaluates each of the \( \text{integer expressions} \) as described for \IntEval. The two results are then compared using the \( \text{relation} \):

\[
\begin{align*}
\text{Equal} & = \\
\text{Greater than} & > \\
\text{Less than} & < 
\end{align*}
\]

For example

\[
\text{IntCompareTF} \{2\} > \{1\} \{\text{Result\{Greater\}}\} \{\text{Result\{Less\}}\}
\]

7.8 Integer Case Functions

\IntCase\{(test integer expression)\}

{\(\text{intexpr case}_1\) \{\text{code case}_1\}}
{\(\text{intexpr case}_2\) \{\text{code case}_2\}}
\ldots
{\(\text{intexpr case}_n\) \{\text{code case}_n\}}

This function evaluates the \( \text{test integer expression} \) and compares this in turn to each of the \( \text{integer expression cases} \). If the two are equal then the associated \( \text{code} \) is left in the input stream and other cases are discarded.

\IntCaseT\{(test integer expression)\}

{\(\text{intexpr case}_1\) \{\text{code case}_1\}}
{\(\text{intexpr case}_2\) \{\text{code case}_2\}}
\ldots
{\(\text{intexpr case}_n\) \{\text{code case}_n\}}
{\{\text{true code}\}}

This function evaluates the \( \text{test integer expression} \) and compares this in turn to each of the \( \text{integer expression cases} \). If the two are equal then the associated \( \text{code} \) is left in the input stream and other cases are discarded. If any of the cases are matched, the \( \text{true code} \) is also inserted into the input stream (after the code for the appropriate case).

\IntCaseF\{(test integer expression)\}

{\(\text{intexpr case}_1\) \{\text{code case}_1\}}
{\(\text{intexpr case}_2\) \{\text{code case}_2\}}
\ldots
{\(\text{intexpr case}_n\) \{\text{code case}_n\}}
{\{\text{false code}\}}

This function evaluates the \( \text{test integer expression} \) and compares this in turn to each of the \( \text{integer expression cases} \). If the two are equal then the associated \( \text{code} \) is left in the input stream and other cases are discarded. If none match then the \( \text{false code} \) is into the input stream (after the code for the appropriate case). For example
This function evaluates the \textit{test integer expression} and compares this in turn to each of the \textit{integer expression cases}. If the two are equal then the associated \textit{code} is left in the input stream and other cases are discarded. If any of the cases are matched, the \textit{true code} is also inserted into the input stream (after the code for the appropriate case), while if none match then the \textit{false code} is inserted.
Chapter 8

Floating Point Numbers (Fp)

8.1 Constant and Scratch Floating Points

\cZeroFp \cMinusZeroFp
Zero, with either sign.

\cOneFp
One as an fp: useful for comparisons in some places.

\cInfFp \cMinusInfFp
Infinity, with either sign. These can be input directly in a floating point expression as inf and -inf.

\cEFp
The value of the base of the natural logarithm, $e = \exp(1)$.

\cPiFp
The value of $\pi$. This can be input directly in a floating point expression as pi.

\cOneDegreeFp
The value of $1^\circ$ in radians. Multiply an angle given in degrees by this value to obtain a result in radians. Note that trigonometric functions expecting an argument in radians or in degrees are both available. Within floating point expressions, this can be accessed as deg.

\lTmpaFp \lTmpbFp \lTmpcFp \lTmpiFp \lTmpjFp \lTmpkFp
Scratch floating point numbers for local assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

\gTmpaFp \gTmpbFp \gTmpcFp \gTmpiFp \gTmpjFp \gTmpkFp
Scratch floating point numbers for global assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.
8.2 Floating Point Expressions

\texttt{\textbackslash FpEval \{\textit{floating point expression}\}}

Evaluates the \textit{(floating point expression)} and returns the result as a decimal number with no exponent. Leading or trailing zeros may be inserted to compensate for the exponent. Non-significant trailing zeros are trimmed, and integers are expressed without a decimal separator. The values $\pm\infty$ and NaN trigger an “invalid operation” exception. For a tuple, each item is converted using \texttt{\textbackslash FpEval} and they are combined as $\langle (fp_1), (fp_2), \ldots, (fp_n) \rangle$ if $n > 1$ and $\langle (fp_1) \rangle$ or $\langle \rangle$ for fewer items. For example

\begin{verbatim}
\texttt{\textbackslash FpEval \{(1.2+3.4)*(5.6-7.8)/9\}}
\end{verbatim}

\begin{verbatim}
\texttt{-1.1244444444444444}
\end{verbatim}

\texttt{\textbackslash FpMathAdd \{\textit{fpexpr}_1\} \{\textit{fpexpr}_2\}}

Adds $\{\textit{fpexpr}_1\}$ and $\{\textit{fpexpr}_2\}$, and returns the result. For example

\begin{verbatim}
\texttt{\textbackslash FpMathAdd \{2.8\} \{3.7\}}
\texttt{\textbackslash FpMathAdd \{3.8-1\} \{2.7+1\}}
\end{verbatim}

\begin{verbatim}
\texttt{6.5 6.5}
\end{verbatim}

\texttt{\textbackslash FpMathSub \{\textit{fpexpr}_1\} \{\textit{fpexpr}_2\}}

Subtracts $\{\textit{fpexpr}_2\}$ from $\{\textit{fpexpr}_1\}$, and returns the result. For example

\begin{verbatim}
\texttt{\textbackslash FpMathSub \{2.8\} \{3.7\}}
\texttt{\textbackslash FpMathSub \{3.8-1\} \{2.7+1\}}
\end{verbatim}

\begin{verbatim}
\texttt{-0.9 -0.9}
\end{verbatim}

\texttt{\textbackslash FpMathMult \{\textit{fpexpr}_1\} \{\textit{fpexpr}_2\}}

Multiplies $\{\textit{fpexpr}_1\}$ by $\{\textit{fpexpr}_2\}$, and returns the result. For example

\begin{verbatim}
\texttt{\textbackslash FpMathMult \{2.8\} \{3.7\}}
\texttt{\textbackslash FpMathMult \{3.8-1\} \{2.7+1\}}
\end{verbatim}

\begin{verbatim}
\texttt{10.36 10.36}
\end{verbatim}

\texttt{\textbackslash FpMathDiv \{\textit{fpexpr}_1\} \{\textit{fpexpr}_2\}}

Divides $\{\textit{fpexpr}_1\}$ by $\{\textit{fpexpr}_2\}$, and returns the result. For example

\begin{verbatim}
\texttt{\textbackslash FpMathDiv \{2.8\} \{3.7\}}
\texttt{\textbackslash FpMathDiv \{3.8-1\} \{2.7+1\}}
\end{verbatim}

\begin{verbatim}
\texttt{0.7567567567567568 0.7567567567567568}
\end{verbatim}

\texttt{\textbackslash FpMathSign \{\textit{fpexpr}\}}

Evaluates the \textit{(fpexpr)} and returns the value using \texttt{\textbackslash FpEval sign(\langle result\rangle)}: $+1$ for positive numbers and for $+\infty$, $-1$ for negative numbers and for $-\infty$, $\pm0$ for $\pm0$. If the operand is a tuple or is NaN, then “invalid operation” occurs and the result is 0. For example

\begin{verbatim}
\texttt{\textbackslash FpMathSign \{3.5\}}
\texttt{\textbackslash FpMathSign \{-2.7\}}
\end{verbatim}

\begin{verbatim}
\texttt{1 -1}
\end{verbatim}
\textbf{\texttt{FpMathAbs}} \{⟨floating point expression⟩\}

Evaluates the ⟨floating point expression⟩ as described for \texttt{FpEval} and returns the absolute value. If the argument is ±∞, NaN or a tuple, “invalid operation” occurs. Within floating point expressions, \texttt{abs()} can be used; it accepts ±∞ and NaN as arguments.

\textbf{\texttt{FpMathMax}} \{(⟨fp expression⟩₁) \} \{⟨fp expression⟩₂\} \textbf{\texttt{FpMathMin}} \{(⟨fp expression⟩₁) \} \{⟨fp expression⟩₂\}

Evaluates the ⟨floating point expressions⟩ as described for \texttt{FpEval} and returns the resulting larger (\texttt{max}) or smaller (\texttt{min}) value. If the argument is a tuple, “invalid operation” occurs, but no other case raises exceptions. Within floating point expressions, \texttt{max()} and \texttt{min()} can be used.

### 8.3 Creating and Using Floating Points

\textbf{\texttt{FpNew}} (⟨fp var⟩)

Creates a new ⟨fp var⟩ or raises an error if the name is already taken. The declaration is global. The ⟨fp var⟩ is initially +0.

\textbf{\texttt{FpConst}} (⟨fp var⟩) \{(⟨floating point expression⟩)\}

Creates a new constant ⟨fp var⟩ or raises an error if the name is already taken. The ⟨fp var⟩ is set globally equal to the result of evaluating the ⟨floating point expression⟩. For example

```
\texttt{FpConst \cMyPiFp} \{3.1415926\}
\texttt{FpUse \cMyPiFp} 3.1415926
```

\textbf{\texttt{FpUse}} (⟨fp var⟩)

Recover the value of the ⟨fp var⟩ and returns the value as a decimal number with no exponent.

### 8.4 Viewing Floating Points

\textbf{\texttt{FpLog}} \{(⟨floating point expression⟩)\}

Evaluates the ⟨floating point expression⟩ and writes the result in the log file.

\textbf{\texttt{FpVarLog}} (⟨fp var⟩)

Writes the value of ⟨fp var⟩ in the log file.

\textbf{\texttt{FpShow}} \{(⟨floating point expression⟩)\}

Evaluates the ⟨floating point expression⟩ and displays the result in the terminal.

\textbf{\texttt{FpVarShow}} (⟨fp var⟩)

Displays the value of ⟨fp var⟩ in the terminal.
8.5 Setting Floating Point Variables

\FpSet \langle fp \ var \rangle \{\langle floating \ point \ expression \rangle\}

Sets \(\langle fp \ var \rangle\) equal to the result of computing the \(\langle floating \ point \ expression \rangle\). For example

\begin{verbatim}
\FpSet \lTmpaFp \{4/7\}
\FpUse \lTmpaFp
\end{verbatim}

0.5714285714285714

\FpSetEq \langle fp \ var_1 \rangle \langle fp \ var_2 \rangle

Sets the floating point variable \(\langle fp \ var_1 \rangle\) equal to the current value of \(\langle fp \ var_2 \rangle\).

\FpZero \langle fp \ var \rangle

Sets the \(\langle fp \ var \rangle\) to +0. For example

\begin{verbatim}
\FpSet \lTmpaFp \{5.3\}
\FpZero \lTmpaFp
\FpUse \lTmpaFp
\end{verbatim}

0

\FpZeroNew \langle fp \ var \rangle

Ensures that the \(\langle fp \ var \rangle\) exists globally by applying \FpNew if necessary, then applies \FpZero to leave the \(\langle fp \ var \rangle\) set to +0.

\FpAdd \langle fp \ var \rangle \{\langle floating \ point \ expression \rangle\}

Adds the result of computing the \(\langle floating \ point \ expression \rangle\) to the \(\langle fp \ var \rangle\). This also applies if \(\langle fp \ var \rangle\) and \(\langle floating \ point \ expression \rangle\) evaluate to tuples of the same size. For example

\begin{verbatim}
\FpSet \lTmpaFp \{5.3\}
\FpAdd \lTmpaFp \{2.11\}
\FpUse \lTmpaFp
\end{verbatim}

7.41

\FpSub \langle fp \ var \rangle \{\langle floating \ point \ expression \rangle\}

Subtracts the result of computing the \(\langle floating \ point \ expression \rangle\) from the \(\langle fp \ var \rangle\). This also applies if \(\langle fp \ var \rangle\) and \(\langle floating \ point \ expression \rangle\) evaluate to tuples of the same size. For example

\begin{verbatim}
\FpSet \lTmpaFp \{5.3\}
\FpSub \lTmpaFp \{2.11\}
\FpUse \lTmpaFp
\end{verbatim}

3.19

8.6 Floating Point Step Functions

\FpStepInline \{\langle initial \ value \rangle\} \{\langle step \rangle\} \{\langle final \ value \rangle\} \{\langle code \rangle\}

This function first evaluates the \(\langle initial \ value \rangle\), \(\langle step \rangle\) and \(\langle final \ value \rangle\), all of which should be floating point expressions evaluating to a floating point number, not a tuple. Then for each \(\langle value \rangle\) from the
\langle \text{initial value} \rangle \text{ to the } \langle \text{final value} \rangle \text{ in turn (using } \langle \text{step} \rangle \text{ between each } \langle \text{value} \rangle \rangle, \text{ the } \langle \text{code} \rangle \text{ is inserted into the input stream with } #1 \text{ replaced by the current } \langle \text{value} \rangle. \text{ Thus the } \langle \text{code} \rangle \text{ should define a function of one argument } (#1). \text{ For example}

\begin{verbatim}
\IgnoreSpacesOn
\TIClear \lTmpaTl
\FpStepInline {1} {0.1} {1.5} {
\TIPutRight \lTmpaTl {[#1]}
}
\Result {\Value\lTmpaTl}
\IgnoreSpacesOff
\end{verbatim}

produces [1][1.1][1.2][1.3][1.4][1.5].

\section{8.7 Float Point Conditionals}

\FpIfExist \langle \text{fp var} \rangle \FpIfExistTF \langle \text{fp var} \rangle {\langle \text{true code} \rangle} {\langle \text{false code} \rangle}

Tests whether the \langle \text{fp var} \rangle is currently defined. \text{ This does not check that the } \langle \text{fp var} \rangle \text{ really is a floating point variable. For example}

\begin{verbatim}
\FpIfExistTF \lTmpaFp {\Result{Yes}} {\Result{No}}
\FpIfExistTF \lMyUndefinedFp {\Result{Yes}} {\Result{No}}
\end{verbatim}

produces \text{Yes No}.

\FpCompare \langle \text{fpexpr}_1 \rangle \langle \text{relation} \rangle \langle \text{fpexpr}_2 \rangle \FpCompareTF \langle \text{fpexpr}_1 \rangle \langle \text{relation} \rangle \langle \text{fpexpr}_2 \rangle {\langle \text{true code} \rangle} {\langle \text{false code} \rangle}

Compares the \langle \text{fpexpr}_1 \rangle \text{ and the } \langle \text{fpexpr}_2 \rangle, \text{ and returns } \text{true} \text{ if the } \langle \text{relation} \rangle \text{ is obeyed. For example}

\begin{verbatim}
\FpCompareTF {1} > {0.9999} {\Result{Greater}} {\Result{Less}}
\FpCompareTF {1} > {1.0001} {\Result{Greater}} {\Result{Less}}
\end{verbatim}

produces \text{Greater Less}.

Two floating points \( x \) and \( y \) may obey four mutually exclusive relations: \( x < y \), \( x = y \), \( x > y \), or \( x \approx y \) ("not ordered"). The last case occurs exactly if one or both operands is NaN or is a tuple, unless they are equal tuples. \text{Note that a NaN is distinct from any value, even another NaN, hence } x = x \text{ is not true for a NaN. To test if a value is NaN, compare it to an arbitrary number with the "not ordered" relation.}

Tuples are equal if they have the same number of items and items compare equal (in particular there must be no NaN). \text{At present any other comparison with tuples yields } ? \text{ (not ordered). This is experimental.
Chapter 9

Dimensions (Dim)

9.1 Constant and Scratch Dimensions

\cMaxDim
The maximum value that can be stored as a dimension. This can also be used as a component of a skip.

\cZeroDim
A zero length as a dimension. This can also be used as a component of a skip.

\lTmpaDim \lTmpbDim \lTmpcDim \lTmpiDim \lTmpjDim \lTmpkDim
Scratch dimensions for local assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

\gTmpaDim \gTmpbDim \gTmpcDim \gTmpiDim \gTmpjDim \gTmpkDim
Scratch dimensions for global assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

9.2 Dimension Expressions

\DimEval \{\langle dimension expression\rangle\}
Evaluates the \langle dimension expression\rangle, expanding any dimensions and token list variables within the \langle expression\rangle to their content (without requiring \DimUse/\TlUse) and applying the standard mathematical rules. The result of the calculation is returned as a \langle dimension denotation\rangle. For example

\DimEval \{(1.2pt+3.4pt)/9\} \quad 0.51111pt

\DimMathAdd \{\langle dimexpr_1\rangle\} \{\langle dimexpr_2\rangle\}
Adds \langle dimexpr_1\rangle and \langle dimexpr_2\rangle, and returns the result. For example

\DimMathAdd \{2.8pt\} \{3.7pt\} \quad 6.5pt 6.5pt
\DimMathAdd \{3.8pt-1pt\} \{2.7pt+1pt\}
\texttt{\textbackslash DimMathSub \{\langle dimexpr\rangle\} \{\langle dimexpr\rangle\}}

Subtracts \{\langle dimexpr\rangle\} from \{\langle dimexpr\rangle\}, and returns the result. For example

\begin{itemize}
  \item \texttt{\textbackslash DimMathSub \{2.8pt\} \{3.7pt\}}
  \item \texttt{\textbackslash DimMathSub \{3.8pt-1pt\} \{2.7pt+1pt\}}
\end{itemize}

The returned value is suitable for use inside a \texttt{\langle dimension expression\rangle} such as

\texttt{\textbackslash DimSet \lTmpaDim \{10pt*\textbackslash DimMathRatio\{5pt\}\{10pt\}}}.

\texttt{\textbackslash DimMathRatio \{\langle dimexpr\rangle\} \{\langle dimexpr\rangle\}}

Parses the two \texttt{\langle dimension expressions\rangle}, then calculates the ratio of the two and returns it. The result is a ratio expression between two integers, with all distances converted to scaled points. For example

\begin{itemize}
  \item \texttt{\textbackslash DimMathRatio \{5pt\} \{10pt\}}
\end{itemize}

The returned value is suitable for use inside a \texttt{\langle dimension expression\rangle} such as

\texttt{\textbackslash DimSet \lTmpaDim \{10pt*\textbackslash DimMathRatio\{5pt\}\{10pt\}}}.

\texttt{\textbackslash DimMathSign \{\langle dimexpr\rangle\}}

Evaluates the \texttt{\langle dimexpr\rangle} then returns 1 or 0 or \(-1\) according to the sign of the result. For example

\begin{itemize}
  \item \texttt{\textbackslash DimMathSign \{3.5pt\}}
  \item \texttt{\textbackslash DimMathSign \{-2.7pt\}}
\end{itemize}

\texttt{\textbackslash DimMathAbs \{\langle dimexpr\rangle\}}

Converts the \texttt{\langle dimexpr\rangle} to its absolute value, returning the result as a \texttt{\langle dimension denotation\rangle}. For example

\begin{itemize}
  \item \texttt{\textbackslash DimMathAbs \{3.5pt\}}
  \item \texttt{\textbackslash DimMathAbs \{-2.7pt\}}
\end{itemize}

\texttt{\textbackslash DimMathMax \{\langle dimexpr\rangle\} \{\langle dimexpr\rangle\}}
\texttt{\textbackslash DimMathMin \{\langle dimexpr\rangle\} \{\langle dimexpr\rangle\}}

Evaluates the two \texttt{\langle dimension expressions\rangle} and returns either the maximum or minimum value as appropriate as a \texttt{\langle dimension denotation\rangle}. For example

\begin{itemize}
  \item \texttt{\textbackslash DimMathMax \{3.5pt\} \{-2.7pt\}}
  \item \texttt{\textbackslash DimMathMin \{3.5pt\} \{-2.7pt\}}
\end{itemize}

9.3 Creating and Using Dimensions

\texttt{\textbackslash DimNew \langle dimension\rangle}

Creates a new \texttt{\langle dimension\rangle} or raises an error if the name is already taken. The declaration is global. The \texttt{\langle dimension\rangle} is initially equal to 0 pt.
CHAPTER 9. DIMENSIONS (\texttt{DIM})

\texttt{\DimConst\{\texttt{\textit{dimension}}\} \{\texttt{(dimension expression)}\}}\hspace{1cm}

Creates a new constant \texttt{(dimension)} or raises an error if the name is already taken. The value of the \texttt{(dimension)} is set globally to the \texttt{(dimension expression)}. For example

\begin{verbatim}
\DimConst \cFooSomeDim \{1cm\}
\DimUse \cFooSomeDim 28.45274pt
\end{verbatim}

\texttt{\DimUse\{\texttt{\textit{dimension}}\}}

Recovery the content of a \texttt{(dimension)} and returns the value. An error is raised if the variable does not exist or if it is invalid.

\subsection*{9.4 Viewing Dimensions}

\texttt{\DimLog\{\texttt{(dimension expression)}\}}

Writes the result of evaluating the \texttt{(dimension expression)} in the log file. For example

\begin{verbatim}
\DimLog \{\lFooSomeDim+1cm\}
\end{verbatim}

\texttt{\DimVarLog\{\texttt{\textit{dimension}}\}}

Writes the value of the \texttt{(dimension)} in the log file. For example

\begin{verbatim}
\DimVarLog \lFooSomeDim
\end{verbatim}

\texttt{\DimShow\{\texttt{(dimension expression)}\}}

Displays the result of evaluating the \texttt{(dimension expression)} on the terminal. For example

\begin{verbatim}
\DimShow \{\lFooSomeDim+1cm\}
\end{verbatim}

\texttt{\DimVarShow\{\texttt{\textit{dimension}}\}}

Displays the value of the \texttt{(dimension)} on the terminal. For example

\begin{verbatim}
\DimVarShow \lFooSomeDim
\end{verbatim}

\subsection*{9.5 Setting Dimension Variables}

\texttt{\DimSet\{\texttt{\textit{dimension}}\} \{\texttt{(dimension expression)}\}}

Sets \texttt{(dimension)} to the value of \texttt{(dimension expression)}, which must evaluate to a length with units.

\texttt{\DimSetEq\{\texttt{\textit{dimension}1}\} \{\texttt{\textit{dimension}2}\}}

Sets the content of \texttt{(dimension}1\} equal to that of \texttt{(dimension}2\}. For example
\DimSet \lTmaDim {10pt}
\DimSetEq \lTmbDim \lTmaDim
\DimUse \lTmbDim

\DimZero \langle \text{dimension} \rangle

Sets \langle \text{dimension} \rangle to 0 pt. For example
\DimSet \lTmaDim {1em}
\DimZero \lTmaDim
\DimUse \lTmaDim

\DimZeroNew \langle \text{dimension} \rangle

Ensures that the \langle \text{dimension} \rangle exists globally by applying \DimNew if necessary, then applies \DimZero to set the \langle \text{dimension} \rangle to zero. For example
\DimZeroNew \lFooSomeDim
\DimUse \lFooSomeDim

\DimAdd \langle \text{dimension} \rangle \{ \langle \text{dimension expression} \rangle \}

Adds the result of the \langle \text{dimension expression} \rangle to the current content of the \langle \text{dimension} \rangle. For example
\DimSet \lTmaDim {5.3pt}
\DimAdd \lTmaDim {2.11pt}
\DimUse \lTmaDim

\DimSub \langle \text{dimension} \rangle \{ \langle \text{dimension expression} \rangle \}

Subtracts the result of the \langle \text{dimension expression} \rangle from the current content of the \langle \text{dimension} \rangle. For example
\DimSet \lTmaDim {5.3pt}
\DimSub \lTmaDim {2.11pt}
\DimUse \lTmaDim

9.6 Dimension Step Functions

\DimStepInline \{ \langle \text{initial value} \rangle \} \{ \langle \text{step} \rangle \} \{ \langle \text{final value} \rangle \} \{ \langle \text{code} \rangle \}

This function first evaluates the \langle \text{initial value} \rangle, \langle \text{step} \rangle and \langle \text{final value} \rangle, all of which should be dimension expressions. Then for each \langle \text{value} \rangle from the \langle \text{initial value} \rangle to the \langle \text{final value} \rangle in turn (using \langle \text{step} \rangle between each \langle \text{value} \rangle), the \langle \text{code} \rangle is inserted into the input stream with #1 replaced by the current \langle \text{value} \rangle. Thus the \langle \text{code} \rangle should define a function of one argument (#1). For example
produces [1.0pt][1.1pt][1.20001pt][1.30002pt][1.40002pt].

This function first evaluates the \(\text{initial value}\), \(\text{step}\) and \(\text{final value}\), all of which should be dimension expressions. Then for each \(\text{value}\) from the \(\text{initial value}\) to the \(\text{final value}\) in turn (using \(\text{step}\) between each \(\text{value}\)), the \(\text{code}\) is inserted into the input stream, with the \(\text{tl var}\) defined as the current \(\text{value}\). Thus the \(\text{code}\) should make use of the \(\text{tl var}\).

### 9.7 Dimension Conditionals

\(\text{\textbackslash DimIfExist } \langle \text{dimension} \rangle\)

\(\text{\textbackslash DimIfExistTF } \langle \text{dimension} \rangle \{\langle \text{true code} \rangle\} \{\langle \text{false code} \rangle\}\)

Tests whether the \(\langle \text{dimension} \rangle\) is currently defined. This does not check that the \(\langle \text{dimension} \rangle\) really is a dimension variable. For example

\[
\text{\textbackslash DimIfExistTF } \\text{lTmpaDim } \{\langle \text{ResultYes} \rangle\} \{\langle \text{ResultNo} \rangle\}
\]

\[
\text{\textbackslash DimIfExistTF } \\text{lFooUndefinedDim } \{\langle \text{ResultYes} \rangle\} \{\langle \text{ResultNo} \rangle\}
\]

\(\text{\textbackslash DimCompare } \{\langle \text{dimexpr} _1 \rangle\} \langle \text{relation} \rangle \{\langle \text{dimexpr} _2 \rangle\}\)

\(\text{\textbackslash DimCompareTF } \{\langle \text{dimexpr} _1 \rangle\} \langle \text{relation} \rangle \{\langle \text{dimexpr} _2 \rangle\} \{\langle \text{true code} \rangle\} \{\langle \text{false code} \rangle\}\)

This function first evaluates each of the \(\langle \text{dimension expressions} \rangle\) as described for \texttt{\textbackslash DimEval}. The two results are then compared using the \(\langle \text{relation} \rangle\):

- \(\text{Equal} =\)
- \(\text{Greater than} >\)
- \(\text{Less than} <\)

For example

\[
\text{\textbackslash DimCompareTF } \{\textit{1pt}\} > \{0.99999pt\} \{\langle \text{ResultGreater} \rangle\} \{\langle \text{ResultLess} \rangle\}
\]

\[
\text{\textbackslash DimCompareTF } \{\textit{1pt}\} > \{1.0001pt\} \{\langle \text{ResultGreater} \rangle\} \{\langle \text{ResultLess} \rangle\}
\]
9.8 Dimension Case Functions

\DimCase {⟨test dimension expression⟩}
{ ⟨dimexpr case1⟩} {⟨code case1⟩}
{ ⟨dimexpr case2⟩} {⟨code case2⟩}
...  {⟨dimexpr case_n⟩} {⟨code case_n⟩}
}

This function evaluates the ⟨test dimension expression⟩ and compares this in turn to each of the ⟨dimension expression cases⟩. If the two are equal then the associated ⟨code⟩ is left in the input stream and other cases are discarded.

\DimCaseT {⟨test dimension expression⟩}
{ ⟨dimexpr case1⟩} {⟨code case1⟩}
{ ⟨dimexpr case2⟩} {⟨code case2⟩}
...  {⟨dimexpr case_n⟩} {⟨code case_n⟩}
{ ⟨true code⟩}

This function evaluates the ⟨test dimension expression⟩ and compares this in turn to each of the ⟨dimension expression cases⟩. If the two are equal then the associated ⟨code⟩ is left in the input stream and other cases are discarded. If any of the cases are matched, the ⟨true code⟩ is also inserted into the input stream (after the code for the appropriate case).

\DimCaseF {⟨test dimension expression⟩}
{ ⟨dimexpr case1⟩} {⟨code case1⟩}
{ ⟨dimexpr case2⟩} {⟨code case2⟩}
...  {⟨dimexpr case_n⟩} {⟨code case_n⟩}
{ ⟨false code⟩}

This function evaluates the ⟨test dimension expression⟩ and compares this in turn to each of the ⟨dimension expression cases⟩. If the two are equal then the associated ⟨code⟩ is left in the input stream and other cases are discarded. If none of the cases match then the ⟨false code⟩ is inserted. For example

\IgnoreSpacesOn
\DimSet \lTmpaDim {5pt}
\DimCaseF {2\lTmpaDim} {5pt} {\Result{Small}}
{4pt+6pt} {\Result{Medium}}
{-10pt} {\Result{Negative}}
{\Result {No Match}}
\IgnoreSpacesOff

Medium
This function evaluates the \textit{(test dimension expression)} and compares this in turn to each of the \textit{(dimension expression cases)}. If the two are equal then the associated \textit{(code)} is left in the input stream and other cases are discarded. If any of the cases are matched, the \textit{(true code)} is also inserted into the input stream (after the code for the appropriate case), while if none match then the \textit{(false code)} is inserted.
Chapter 10

Comma Separated Lists (Clist)

10.1 Constant and Scratch Comma Lists

\cEmptyClist

Constant that is always empty.

\lTmpaClist \lTmpbClist \lTmpcClist \lTmpiClist \lTmpjClist \lTmpkClist

Scratch comma lists for local assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

\gTmpaClist \gTmpbClist \gTmpcClist \gTmpiClist \gTmpjClist \gTmpkClist

Scratch comma lists for global assignment. These are never used by the functional package, and so are safe for use with any function. However, they may be overwritten by other code and so should only be used for short-term storage.

10.2 Creating and Using Comma Lists

\ClistNew ⟨comma list⟩

Creates a new ⟨comma list⟩ or raises an error if the name is already taken. The declaration is global. The ⟨comma list⟩ initially contains no items.

\ClistConst ⟨clist var⟩ {⟨comma list⟩}

Creates a new constant ⟨clist var⟩ or raises an error if the name is already taken. The value of the ⟨clist var⟩ is set globally to the ⟨comma list⟩.

\ClistVarJoin ⟨clist var⟩ {⟨separator⟩}

Returns the contents of the ⟨clist var⟩, with the ⟨separator⟩ between the items. For example,

\ClistSet \lTmpaClist { a , b , , c , {de} , f }
\ClistVarJoin \lTmpaClist { and }

a and b and c and de and f
CHAPTER 10. COMMA SEPARATED LISTS (CLIST)

\ClistVarJoinExtended ⟨clist var⟩ {⟨separator between two⟩} {⟨separator between more than two⟩} {⟨separator between final two⟩}

Returns the contents of the ⟨clist var⟩, with the appropriate ⟨separator⟩ between the items. Namely, if the comma list has more than two items, the ⟨separator between more than two⟩ is placed between each pair of items except the last, for which the ⟨separator between final two⟩ is used. If the comma list has exactly two items, then they are joined with the ⟨separator between two⟩ and returns. For example,

\ClistSet \lTmpaClist { a , b , , c , {de} , f }
\ClistVarJoinExtended \lTmpaClist { and } {, } {, and }
a, b, c, de, and f

The first separator argument is not used in this case because the comma list has more than 2 items.

\ClistJoin ⟨comma list⟩ {⟨separator⟩}
\ClistJoinExtended ⟨comma list⟩ {⟨separator between two⟩} {⟨separator between more than two⟩} {⟨separator between final two⟩}

Returns the contents of the ⟨comma list⟩, with the appropriate ⟨separator⟩ between the items. As for \ClistSet, blank items are omitted, spaces are removed from both sides of each item, then a set of braces is removed if the resulting space-trimmed item is braced. The ⟨separators⟩ are then inserted in the same way as for \ClistJoin and \ClistJoinExtended, respectively.

10.3 Viewing Comma Lists

\ClistLog {⟨tokens⟩}

Writes the entries in the comma list in the log file. See also \ClistShow which displays the result in the terminal.

\ClistVarLog ⟨comma list⟩

Writes the entries in the ⟨comma list⟩ in the log file. See also \ClistVarShow which displays the result in the terminal.

\ClistShow {⟨tokens⟩}

Displays the entries in the comma list in the terminal.

\ClistVarShow ⟨comma list⟩

Displays the entries in the ⟨comma list⟩ in the terminal.

10.4 Setting Comma Lists

\ClistSet ⟨comma list⟩ {⟨item₁},…,⟨itemₙ⟩}

Sets ⟨comma list⟩ to contain the ⟨items⟩, removing any previous content from the variable. Blank items are omitted, spaces are removed from both sides of each item, then a set of braces is removed if the resulting space-trimmed item is braced. To store some ⟨tokens⟩ as a single ⟨item⟩ even if the ⟨tokens⟩ contain commas or spaces, add a set of braces: \ClistSet ⟨comma list⟩ { ⟨{⟨tokens⟩}⟩}. 
\ClistSetEq \langle \text{comma list}_1 \rangle \langle \text{comma list}_2 \rangle

Sets the content of \langle \text{comma list}_1 \rangle equal to that of \langle \text{comma list}_2 \rangle. To set a token list variable equal to a comma list variable, use \TlSetEq. Conversely, setting a comma list variable to a token list is unadvisable unless one checks space-trimming and related issues.

\ClistClear \langle \text{comma list} \rangle

Clears all items from the \langle \text{comma list} \rangle.

\ClistClearNew \langle \text{comma list} \rangle

Ensures that the \langle \text{comma list} \rangle exists globally by applying \ClistNew if necessary, then applies \ClistClear to leave the list empty.

\ClistConcat \langle \text{comma list}_1 \rangle \langle \text{comma list}_2 \rangle \langle \text{comma list}_3 \rangle

Concatenates the content of \langle \text{comma list}_2 \rangle and \langle \text{comma list}_3 \rangle together and saves the result in \langle \text{comma list}_1 \rangle. The items in \langle \text{comma list}_2 \rangle are placed at the left side of the new comma list.

\ClistPutLeft \langle \text{comma list} \rangle \{ \langle \text{item}_1 \rangle, ..., \langle \text{item}_n \rangle \}

Appends the \langle \text{items} \rangle to the left of the \langle \text{comma list} \rangle. Blank items are omitted, spaces are removed from both sides of each item, then a set of braces is removed if the resulting space-trimmed item is braced. To append some \langle \text{tokens} \rangle as a single \langle \text{item} \rangle even if the \langle \text{tokens} \rangle contain commas or spaces, add a set of braces: \ClistPutLeft \langle \text{comma list} \rangle \{ \langle \text{tokens} \rangle \}.

\ClistPutRight \langle \text{comma list} \rangle \{ \langle \text{item}_1 \rangle, ..., \langle \text{item}_n \rangle \}

Appends the \langle \text{items} \rangle to the right of the \langle \text{comma list} \rangle. Blank items are omitted, spaces are removed from both sides of each item, then a set of braces is removed if the resulting space-trimmed item is braced. To append some \langle \text{tokens} \rangle as a single \langle \text{item} \rangle even if the \langle \text{tokens} \rangle contain commas or spaces, add a set of braces: \ClistPutRight \langle \text{comma list} \rangle \{ \langle \text{tokens} \rangle \}.

10.5 Modifying Comma Lists

While comma lists are normally used as ordered lists, it may be necessary to modify the content. The functions here may be used to update comma lists, while retaining the order of the unaffected entries.

\ClistRemoveDuplicates \langle \text{comma list} \rangle

Removes duplicate items from the \langle \text{comma list} \rangle, leaving the left most copy of each item in the \langle \text{comma list} \rangle. The \langle \text{item} \rangle comparison takes place on a token basis, as for \TlIfEqTF. This function iterates through every item in the \langle \text{comma list} \rangle and does a comparison with the \langle \text{items} \rangle already checked. It is therefore relatively slow with large comma lists. Furthermore, it may fail if any of the items in the \langle \text{comma list} \rangle contains \{, \}, or # (assuming the usual \TeX{} category codes apply).

\ClistRemoveAll \langle \text{comma list} \rangle \{ \langle \text{item} \rangle \}

Removes every occurrence of \langle \text{item} \rangle from the \langle \text{comma list} \rangle. The \langle \text{item} \rangle comparison takes place on a token basis, as for \TlIfEqTF. The function may fail if the \langle \text{item} \rangle contains \{, \}, or # (assuming the usual \TeX{} category codes apply).
CHAPTER 10. COMMA SEPARATED LISTS (CLIST)

\ClistReverse (comma list)

Reverses the order of items stored in the (comma list).

10.6 Working with the Contents of Comma Lists

\ClistCount {(comma list)\}
\ClistVarCount (comma list)

Returns the number of items in the (comma list) as an (integer denotation). The total number of items in a (comma list) includes those which are duplicates, i.e. every item in a (comma list) is counted.

\ClistItem {(comma list)\} {(integer expression)\}

Indexing items in the (comma list) from 1 at the top (left), this function evaluates the (integer expression) and returns the appropriate item from the comma list. If the (integer expression) is negative, indexing occurs from the bottom (right) of the comma list. When the (integer expression) is larger than the number of items in the (comma list) (as calculated by \ClistCount) then the function returns nothing.

\ClistVarItem (comma list) {(integer expression)\}

Indexing items in the (comma list) from 1 at the top (left), this function evaluates the (integer expression) and returns the appropriate item from the comma list. If the (integer expression) is negative, indexing occurs from the bottom (right) of the comma list. When the (integer expression) is larger than the number of items in the (comma list) (as calculated by \ClistVarCount) then the function returns nothing.

\ClistRandItem {(comma list)\}
\ClistVarRandItem (clist var)

Selects a pseudo-random item of the (comma list). If the (comma list) has no item, the result is empty.

10.7 Comma Lists as Stacks

Comma lists can be used as stacks, where data is pushed to and popped from the top of the comma list. (The left of a comma list is the top, for performance reasons.) The stack functions for comma lists are not intended to be mixed with the general ordered data functions detailed in the previous section: a comma list should either be used as an ordered data type or as a stack, but not in both ways.

\ClistGet (comma list) (token list variable)
\ClistGetT (comma list) (token list variable) (true code)
\ClistGetF (comma list) (token list variable) (false code)
\ClistGetTF (comma list) (token list variable) (true code) (false code)

Stores the left-most item from a (comma list) in the (token list variable) without removing it from the (comma list). The (token list variable) is assigned locally.

\ClistPop (comma list) (token list variable)
\ClistPopT (comma list) (token list variable) {(true code)\}
\ClistPopF (comma list) (token list variable) {(false code)\}
\ClistPopTF (comma list) (token list variable) {(true code)} {(false code)\}

Pops the left-most item from a (comma list) into the (token list variable), i.e. removes the item from the comma list and stores it in the (token list variable). The assignment of the (token list variable) is local.
If the ⟨comma list⟩ is empty, the value of the ⟨token list variable⟩ is not defined in this case and should not be relied upon.

\ClistPush ⟨comma list⟩ {⟨items⟩}

Adds the {⟨items⟩} to the top of the ⟨comma list⟩. Spaces are removed from both sides of each item as for any n-type comma list.

### 10.8 Mapping over Comma Lists

When the comma list is given explicitly, spaces are trimmed around each item. If the result of trimming spaces is empty, the item is ignored. Otherwise, if the item is surrounded by braces, one set is removed, and the result is passed to the mapped function. Thus, if the comma list that is being mapped is \{a, \{b\}, c, \}, then the arguments passed to the mapped function are ‘a’, ‘\{b\}’, an empty argument, and ‘c’.

When the comma list is given as a variable, spaces have already been trimmed on input, and items are simply stripped of one set of braces if any. This case is more efficient than using explicit comma lists.

\ClistMapInline {⟨comma list⟩} {⟨inline function⟩}
\ClistVarMapInline ⟨comma list⟩ {⟨inline function⟩}

Applies ⟨inline function⟩ to every ⟨item⟩ stored within the ⟨comma list⟩. The ⟨inline function⟩ should consist of code which receives the ⟨item⟩ as \#1. The ⟨items⟩ are returned from left to right. For example

\IgnoreSpacesOn
\TLClear \lTmpaTl
\ClistMapInline {one,two,three} {
   \TLPutRight \lTmpaTl {(#1)}
}
\Result {\TLUse\lTmpaTl}
\IgnoreSpacesOff

produces (one)(two)(three).

\ClistMapVariable {⟨comma list⟩} ⟨variable⟩ {⟨code⟩}
\ClistVarMapVariable ⟨comma list⟩ ⟨variable⟩ {⟨code⟩}

Stores each ⟨item⟩ of the ⟨comma list⟩ in turn in the (token list) ⟨variable⟩ and applies the ⟨code⟩. The ⟨code⟩ will usually make use of the ⟨variable⟩, but this is not enforced. The assignments to the ⟨variable⟩ are local. Its value after the loop is the last ⟨item⟩ in the ⟨comma list⟩, or its original value if there were no ⟨item⟩. The ⟨items⟩ are returned from left to right.

### 10.9 Comma List Conditionals

\ClistIfExist ⟨comma list⟩
\ClistIfExistTF ⟨comma list⟩ {⟨true code⟩} {⟨false code⟩}

Tests whether the ⟨comma list⟩ is currently defined. This does not check that the ⟨comma list⟩ really is a comma list.

\ClistIfEmpty ⟨comma list⟩
\ClistIfEmptyTF ⟨comma list⟩ {⟨true code⟩} {⟨false code⟩}

Tests if the ⟨comma list⟩ is empty (containing no items). The rules for space trimming are as for other
n-type comma-list functions, hence the comma list \{ , , , \} (without outer braces) is empty, while
\{ ,{},\} (without outer braces) contains one element, which happens to be empty: the comma-list is not
empty.

\ClistVarIfEmpty \{comma list\}
\ClistVarIfEmptyTF \{comma list\} \{(true code)\} \{(false code)\}

Tests if the \{comma list\} is empty (containing no items).

\ClistIfIn \{\{comma list\}\} \{\{item\}\}
\ClistIfInTF \{\{comma list\}\} \{\{item\}\} \{(true code)\} \{(false code)\}

Tests if the \{item\} is present in the \{comma list\}. In the case of an n-type \{comma list\}, the usual rules
of space trimming and brace stripping apply. For example

\ClistIfInTF \{ a , {b} , {b} , c \} \{b\} \{Yes\} \{No\} Yes

\ClistVarIfIn \{comma list\} \{\{item\}\}
\ClistVarIfInTF \{comma list\} \{\{item\}\} \{(true code)\} \{(false code)\}

Tests if the \{item\} is present in the \{comma list\}. In the case of an n-type \{comma list\}, the usual rules
of space trimming and brace stripping apply.
Chapter 11

The Source Code

11.1 Interfaces for Functional Programming (Prg)

\NeedsTeXFormat{LaTeX2e}[2018-04-01]

\RequirePackage{expl3}
\ProvidesExplPackage{functional}{2022-03-19}{2022B}
{^^JLaTeX2 functional interfaces for LaTeX3 programming layer}

\cs_generate_variant:Nn \iow_log:n { V }
\cs_generate_variant:Nn \tl_log:n { e }
\cs_generate_variant:Nn \tl_set:Nn { Ne }
\tl_new:N \gResultTl
\int_new:N \l__fun_arg_count_int
\tl_new:N \l__fun_parameters_defined_tl
\tl_const:Nn \c__fun_parameter_defined_i_i_tl { #1 } % no argument
\tl_const:Nn \c__fun Parameter defined_i_i_tl { #1 }
\tl_const:Nn \c__fun Parameter defined_i_ii_tl { #1 #2 }
\tl_const:Nn \c__fun Parameter defined_i_iii_tl { #1 #2 #3 }
\tl_const:Nn \c__fun Parameter defined_i_iv_tl { #1 #2 #3 #4 }
\tl_const:Nn \c__fun Parameter defined_i_v_tl { #1 #2 #3 #4 #5 }
\tl_const:Nn \c__fun Parameter defined_i_vi_tl { #1 #2 #3 #4 #5 #6 }
\tl_const:Nn \c__fun Parameter defined_i_vii_tl { #1 #2 #3 #4 #5 #6 #7 }
\tl_const:Nn \c__fun Parameter defined_i_viii_tl { #1 #2 #3 #4 #5 #6 #7 #8 }
\tl_const:Nn \c__fun Parameter defined_i_ix_tl { #1 #2 #3 #4 #5 #6 #7 #8 #9 }
\tl_new:N \l__fun_parameters called_tl
\tl_const:Nn \c__fun Parameter called_i_i_tl { #1 } % no argument
\tl_const:Nn \c__fun Parameter called_i_i_tl { #1}{#2} % no argument
\tl_const:Nn \c__fun Parameter called_i_ii_tl { #1}{#2}{#3} % no argument
\tl_const:Nn \c__fun Parameter called_i_iii_tl { #1}{#2}{#3}{#4} % no argument
\tl_const:Nn \c__fun Parameter called_i_iv_tl { #1}{#2}{#3}{#4}{#5} % no argument
\tl_const:Nn \c__fun Parameter called_i_v_tl { #1}{#2}{#3}{#4}{#5}{#6} % no argument
\tl_const:Nn \c__fun Parameter called_i_vi_tl { #1}{#2}{#3}{#4}{#5}{#6}{#7} % no argument
\tl_const:Nn \c__fun Parameter called_i_vii_tl { #1}{#2}{#3}{#4}{#5}{#6}{#7}{#8} % no argument

55
\_tl_new:N \l\_\_fun\_parameters\_true\_tl
\_tl_new:N \l\_\_fun\_parameters\_false\_tl
\tl_const:Nn \c\_\_fun\_parameter\_called\_i\_tl { {#1} }
\tl_const:Nn \c\_\_fun\_parameter\_called\_ii\_tl { {#2} }
\tl_const:Nn \c\_\_fun\_parameter\_called\_iii\_tl { {#3} }
\tl_const:Nn \c\_\_fun\_parameter\_called\_iv\_tl { {#4} }
\tl_const:Nn \c\_\_fun\_parameter\_called\_v\_tl { {#5} }
\tl_const:Nn \c\_\_fun\_parameter\_called\_vi\_tl { {#6} }
\tl_const:Nn \c\_\_fun\_parameter\_called\_vii\_tl { {#7} }
\tl_const:Nn \c\_\_fun\_parameter\_called\_viii\_tl { {#8} }
\tl_const:Nn \c\_\_fun\_parameter\_called\_ix\_tl { {#9} }

%%% #1: function name; #2: argument specification; #3 function body
\cs_new_protected:Npn \_\_fun\_new\_function:Nnn #1 #2 #3
{\int_set:Nn \l\_\_fun\_arg\_count\_int { \tl_count:n {#2} } % spaces are ignored
\tl_set_eq:Nc \l\_\_fun\_parameters\_defined\_tl
{ c\_\_fun\_parameter\_defined\_i\_ \int_to_roman:n { \l\_\_fun\_arg\_count\_int } _tl }
\exp_last_unbraced:NcV \cs_new_protected:Npn
\{ __fun\_defined\_ \cs_to_str:N #1 : w \}
\l\_\_fun\_parameters\_defined\_tl
{ \__fun\_group\_begin:
\tl_gclear:N \g\Result\_Tl
#3
\__fun\_tracing\_log:e { [O] \g\Result\_Tl }
\__fun\_group\_end:
}
\use:c { __fun\_new\_with\_arg\_ \int_to_roman:n { \l\_\_fun\_arg\_count\_int } :NnV }
#1 {#2} \l\_\_fun\_parameters\_defined\_tl
}
\cs_generate_variant:Nn \_\_fun\_new\_function:Nnn { cne }
\cs_set_eq:NN \Prg\New\Function \_\_fun\_new\_function:Nnn
\tl_new:N \g\__fun\_last\_result\_tl

%%% #1: function name; #2: argument specification; #3 function body
\cs_new_protected:Npn \_\_fun\_new\_conditional:Nnn #1 #2 #3
{ \_\_fun\_new\_function:Nnn #1 { #2 } { #3 }
\int_set_eq:Nc \l\_\_fun\_parameters\_called\_tl
{ c\_\_fun\_parameter\_called\_i\_ \int_to_roman:n { \l\_\_fun\_arg\_count\_int } _tl }
\int_set_eq:Nc \l\_\_fun\_parameters\_true\_tl
{ c\_\_fun\_parameter\_called\_ \int_to_roman:n { \l\_\_fun\_arg\_count\_int + 1 } _tl }
\int_set_eq:Nc \l\_\_fun\_parameters\_false\_tl
{ c\_\_fun\_parameter\_called\_ \int_to_roman:n { \l\_\_fun\_arg\_count\_int + 2 } _tl }
\_\_fun\_new\_function:cne \{ \cs_to_str:N #1 TF \} { #2 n n }
{ #1 \exp_not:V \l\_\_fun\_parameters\_called\_tl
\exp_not:n
{ \tl_set_eq:NN \g\__fun\_last\_result\_tl \g\Result\_Tl
\tl_gclear:N \g\Result\_Tl
\exp_last_unbraced:NV \bool_if:NTF \g\__fun\_last\_result\_tl
\exp_not:V \l\_\_fun\_parameters\_true\_tl
}
\exp_not:V \l__fun_parameters_false_tl
}

\cs_set_eq:NN \PrgNewConditional \__fun_new_conditional:Nnn

\int_new:N \g__fun_nesting_level_int

%%% #1: function name; #2: argument specifications; #3 parameters tl defined
%%% Some times we need to create a function without arguments
\cs_new_protected:Npn \__fun_new_with_arg_:Nnn #1 #2 #3
{\cs_new_protected:Npn #1 #3
{\int_gincr:N \g__fun_nesting_level_int
 \__fun_evaluate:Nn #1 {#2}
 \int_gdecr:N \g__fun_nesting_level_int
 \__fun_return_result:
}}
\cs_generate_variant:Nn \__fun_new_with_arg_:Nnn { NnV }

%%% #1: function name; #2: argument specifications; #3 parameters tl defined
\cs_new_protected:Npn \__fun_new_with_arg_i:Nnn #1 #2 #3
{\cs_new_protected:Npn #1 #3
{\int_gincr:N \g__fun_nesting_level_int
 \__fun_one_argument_gset:nn { 1 } { ##1 }
 \__fun_evaluate:Nn #1 {#2}
 \int_gdecr:N \g__fun_nesting_level_int
 \__fun_return_result:
}}
\cs_generate_variant:Nn \__fun_new_with_arg_i:Nnn { NnV }

%%% #1: function name; #2: argument specifications; #3 parameters tl defined
\cs_new_protected:Npn \__fun_new_with_arg_ii:Nnn #1 #2 #3
{\cs_new_protected:Npn #1 #3
{\int_gincr:N \g__fun_nesting_level_int
 \__fun_one_argument_gset:nn { 1 } { ##1 }
 \__fun_one_argument_gset:nn { 2 } { ##2 }
 \__fun_evaluate:Nn #1 {#2}
 \int_gdecr:N \g__fun_nesting_level_int
 \__fun_return_result:
}}
\cs_generate_variant:Nn \__fun_new_with_arg_ii:Nnn { NnV }

%%% #1: function name; #2: argument specifications; #3 parameters tl defined
\cs_new_protected:Npn \__fun_new_with_arg_iii:Nnn #1 #2 #3
{\cs_new_protected:Npn #1 #3
{\int_gincr:N \g__fun_nesting_level_int
 \__fun_evaluate:Nn #1 {#2}
 \int_gdecr:N \g__fun_nesting_level_int
 \__fun_return_result:
}}
\_\_fun\_one\_argument\_gset:nn \{ 1 \} \{ ##1 \}
\_\_fun\_one\_argument\_gset:nn \{ 2 \} \{ ##2 \}
\_\_fun\_one\_argument\_gset:nn \{ 3 \} \{ ##3 \}
\_\_fun\_evaluate:Nn \#1 \{ ##2 \}
\int\_gdecr:N \g\__\_fun\_nesting\_level\_int
\_\_fun\_return\_result:
}
\cs\_generate\_variant:Nn \_\_fun\_new\_with\_arg\_iii:Nnn \{ \texttt{NnV} \}

%% #1: function name; #2: argument specifications; #3 parameters tl defined
\cs\_new\_protected:Npn \_\_fun\_new\_with\_arg\_iv:Nnn \#1 \#2 \#3
{
\cs\_new\_protected:Npn \#1 \#3
{
\int\_gincr:N \g\__\_fun\_nesting\_level\_int
\_\_fun\_one\_argument\_gset:nn \{ 1 \} \{ ##1 \}
\_\_fun\_one\_argument\_gset:nn \{ 2 \} \{ ##2 \}
\_\_fun\_one\_argument\_gset:nn \{ 3 \} \{ ##3 \}
\_\_fun\_one\_argument\_gset:nn \{ 4 \} \{ ##4 \}
\_\_fun\_evaluate:Nn \#1 \{ ##2 \}
\int\_gdecr:N \g\__\_fun\_nesting\_level\_int
\_\_fun\_return\_result:
}
\cs\_generate\_variant:Nn \_\_fun\_new\_with\_arg\_iv:Nnn \{ \texttt{NnV} \}

%% #1: function name; #2: argument specifications; #3 parameters tl defined
\cs\_new\_protected:Npn \_\_fun\_new\_with\_arg\_v:Nnn \#1 \#2 \#3
{
\cs\_new\_protected:Npn \#1 \#3
{
\int\_gincr:N \g\__\_fun\_nesting\_level\_int
\_\_fun\_one\_argument\_gset:nn \{ 1 \} \{ ##1 \}
\_\_fun\_one\_argument\_gset:nn \{ 2 \} \{ ##2 \}
\_\_fun\_one\_argument\_gset:nn \{ 3 \} \{ ##3 \}
\_\_fun\_one\_argument\_gset:nn \{ 4 \} \{ ##4 \}
\_\_fun\_one\_argument\_gset:nn \{ 5 \} \{ ##5 \}
\_\_fun\_evaluate:Nn \#1 \{ ##2 \}
\int\_gdecr:N \g\__\_fun\_nesting\_level\_int
\_\_fun\_return\_result:
}
\cs\_generate\_variant:Nn \_\_fun\_new\_with\_arg\_v:Nnn \{ \texttt{NnV} \}

%% #1: function name; #2: argument specifications; #3 parameters tl defined
\cs\_new\_protected:Npn \_\_fun\_new\_with\_arg\_vi:Nnn \#1 \#2 \#3
{
\cs\_new\_protected:Npn \#1 \#3
{
\int\_gincr:N \g\__\_fun\_nesting\_level\_int
\_\_fun\_one\_argument\_gset:nn \{ 1 \} \{ ##1 \}
\_\_fun\_one\_argument\_gset:nn \{ 2 \} \{ ##2 \}
\_\_fun\_one\_argument\_gset:nn \{ 3 \} \{ ##3 \}
\_\_fun\_one\_argument\_gset:nn \{ 4 \} \{ ##4 \}
\_\_fun\_one\_argument\_gset:nn \{ 5 \} \{ ##5 \}
\_\_fun\_one\_argument\_gset:nn \{ 6 \} \{ ##6 \}
\_\_fun\_evaluate:Nn \#1 \{ ##2 \}
\int_gdecr:N \g__fun_nesting_level_int
\__fun_return_result:
}
\cs_generate_variant:Nn \__fun_new_with_arg_vi:Nnn { NnV }

%%% #1: function name; #2: argument specifications; #3 parameters tl defined
\cs_new_protected:Npn \__fun_new_with_arg_vii:Nnn #1 #2 #3
{
\cs_new_protected:Npn #1 #3
{
\int_gincr:N \g__fun_nesting_level_int
\__fun_one_argument_gset:nn { 1 } { ##1 }
\__fun_one_argument_gset:nn { 2 } { ##2 }
\__fun_one_argument_gset:nn { 3 } { ##3 }
\__fun_one_argument_gset:nn { 4 } { ##4 }
\__fun_one_argument_gset:nn { 5 } { ##5 }
\__fun_one_argument_gset:nn { 6 } { ##6 }
\__fun_one_argument_gset:nn { 7 } { ##7 }
\__fun_evaluate:Nn #1 {#2}
\int_gdecr:N \g__fun_nesting_level_int
\__fun_return_result:
}
}\cs_generate_variant:Nn \__fun_new_with_arg_vii:Nnn { NnV }

%%% #1: function name; #2: argument specifications; #3 parameters tl defined
\cs_new_protected:Npn \__fun_new_with_arg_viii:Nnn #1 #2 #3
{
\cs_new_protected:Npn #1 #3
{
\int_gincr:N \g__fun_nesting_level_int
\__fun_one_argument_gset:nn { 1 } { ##1 }
\__fun_one_argument_gset:nn { 2 } { ##2 }
\__fun_one_argument_gset:nn { 3 } { ##3 }
\__fun_one_argument_gset:nn { 4 } { ##4 }
\__fun_one_argument_gset:nn { 5 } { ##5 }
\__fun_one_argument_gset:nn { 6 } { ##6 }
\__fun_one_argument_gset:nn { 7 } { ##7 }
\__fun_one_argument_gset:nn { 8 } { ##8 }
\__fun_evaluate:Nn #1 {#2}
\int_gdecr:N \g__fun_nesting_level_int
\__fun_return_result:
}
}\cs_generate_variant:Nn \__fun_new_with_arg_viii:Nnn { NnV }

%%% #1: function name; #2: argument specifications; #3 parameters tl defined
\cs_new_protected:Npn \__fun_new_with_arg_ix:Nnn #1 #2 #3
{
\cs_new_protected:Npn #1 #3
{
\int_gincr:N \g__fun_nesting_level_int
\__fun_one_argument_gset:nn { 1 } { ##1 }
\__fun_one_argument_gset:nn { 2 } { ##2 }
\__fun_one_argument_gset:nn { 3 } { ##3 }
\__fun_one_argument_gset:nn { 4 } { ##4 }
\__fun_one_argument_gset:nn { 5 } { ##5 }
\__fun_one_argument_gset:nn { 6 } { \#6 }
\__fun_one_argument_gset:nn { 7 } { \#7 }
\__fun_one_argument_gset:nn { 8 } { \#8 }
\__fun_one_argument_gset:nn { 9 } { \#9 }
\__fun_evaluate:Nn \#1 { \#2 }
\int_gdecr:N \g__fun_nesting_level_int
\__fun_return_result:
}
}
\cs_generate_variant:Nn \__fun_new_with_arg_ix:Nnn { NnV }
\tl_new:N \l__fun_argtype_tl
\tl_const:Nn \c__fun_argtype_m_tl { m }
\tl_const:Nn \c__fun_argtype_M_tl { M }
\tl_const:Nn \c__fun_argtype_n_tl { n }
\tl_const:Nn \c__fun_argtype_N_tl { N }
\tl_new:N \l__fun_argument_tl

%%% #1: function name; #2: argument specifications
\cs_new_protected:Npn \__fun_evaluate:Nn \#1 \#2
{
    \__fun_argtype_index_gzero:
    \__fun_arguments_gclear:
    \tl_map_variable:nNn { \#2 } \l__fun_argtype_tl \% spaces are ignored
    {
        \__fun_argtype_index_gincr:
        \__fun_one_argument_get:eN { \__fun_argtype_index_use: } \l__fun_argument_tl
        \tl_case:Nn \l__fun_argtype_tl
        { \c__fun_argtype_m_tl
            { \__fun_evaluate_and_put_argument:N \l__fun_argument_tl }
        }
        { \c__fun_argtype_M_tl
            { \__fun_evaluate_and_put_argument:N \l__fun_argument_tl }
        }
        { \c__fun_argtype_n_tl
            { \__fun_arguments_gput:e { { \exp_not:V \l__fun_argument_tl } } }
        }
        { \c__fun_argtype_N_tl
            { \__fun_arguments_gput:e { \exp_not:V \l__fun_argument_tl } }
        }
    }
    \__fun_arguments_log:N \#1
    \__fun_arguments_called:c { \__fun_defined_ \cs_to_str:N \#1 : w }
}

\cs_new_protected:Npn \__fun_evaluate_and_put_argument:N \#1
{
    \cs_if_exist:cTF
    { \__fun_defined_ \exp_last_unbraced:Ne \cs_to_str:N { \tl_head:N \#1 } : w }
    {
        \__fun_defined_ \exp_last_unbraced:Ne \cs_to_str:N { \tl_head:N \#1 } : w
    }
#1
\_\_fun_arguments_gput:e { { \exp_not:V \gResultTl } }
}
\__fun_arguments_gput:e { { \exp_not:V #1 } }
}

\_\_fun_one_argument_gset:nn #1 #2
\tl_gset:cn
  \_\_fun_one_argument_ \int_use:N \g__fun_nesting_level_int _#1_tl \{ #2 \}
\_\_fun_one_argument_log:nn \{ \exp_not:V \gResultTl \} { set }

\_\_fun_one_argument_get:nN #1 #2
\tl_set_eq:Nc
  #2 \{ \_\_fun_one_argument_ \int_use:N \g__fun_nesting_level_int _#1_tl \}
\_\_fun_one_argument_log:nn \{ \exp_not:V \gResultTl \} { get }
\_\_fun_one_argument_log:nn #1 #2
\tl_log:e\{ \exp_not:V \gResultTl \}

\_\_fun_one_argument_log:nn #1 #2
\tl_log:e\{ \exp_not:V \gResultTl \}

\int_new:c \{ \_\_fun_argtype_index_1_int \}
\int_new:c \{ \_\_fun_argtype_index_2_int \}
\int_new:c \{ \_\_fun_argtype_index_3_int \}
\int_new:c \{ \_\_fun_argtype_index_4_int \}
\int_new:c \{ \_\_fun_argtype_index_5_int \}

\_\_fun_argtype_index_gzero:\}
\int_gzero_new:c\{ \_\_fun_argtype_index_ \int_use:N \g__fun_nesting_level_int _int \}

\_\_fun_argtype_index_gincr:\}
\int_gincr:c\{ \_\_fun_argtype_index_ \int_use:N \g__fun_nesting_level_int _int \}

\_\_fun_argtype_index_use:\}

\int_use:c { g__fun_argtype_index_ \int_use:N g__fun_nesting_level_int_int } }

\cs_new_protected:Npn \__fun_arguments_called:N #1
{ \exp_last_unbraced:Nv #1 { g__fun_arguments_ \int_use:N g__fun_nesting_level_int_tl } }

\cs_generate_variant:Nn \__fun_arguments_called:N { c }

\cs_new_protected:Npn \__fun_arguments_gclear:
{ \tl_gclear:c { g__fun_arguments_ \int_use:N g__fun_nesting_level_int_tl } }

\cs_new_protected:Npn \__fun_arguments_log:N #1
{ \__fun_tracing_log:e
  { [I] ~ \token_to_str:N #1 \exp_not:v { g__fun_arguments_ \int_use:N g__fun_nesting_level_int_tl } }
}

\cs_new_protected:Npn \__fun_arguments_gput:n #1
{ \tl_gput_right:cn { g__fun_arguments_ \int_use:N g__fun_nesting_level_int_tl } { #1 } }

\cs_generate_variant:Nn \__fun_arguments_gput:n { e }

\cs_set_eq:NN \Break \prg_break:
\cs_set_eq:NN \PrgBreak \prg_break:
\cs_set_eq:NN \BreakDo \prg_break:n
\cs_set_eq:NN \PrgBreakDo \prg_break:n

\cs_new_protected:Npn \__fun_put_result:n #1
{ \tl_gput_right:Nn \gResultTl { #1 } }

\cs_generate_variant:Nn \__fun_put_result:n { e, V }
\PrgNewFunction \Result { m }
{ \__fun_put_result:n { #1 } }
\cs_new_protected:Npn \__fun_return_result:
{ \int_compare:nNnT { \g__fun_nesting_level_int_int } = { 0 } 
  { \tl_use:N \gResultTl } }

\tl_new:N \l__fun_variable_type_tl
\prg_new_protected_conditional:Npnn \__fun_if_global_variable:N #1 { TF }
{\tl_set:Ne \l__fun_variable_type_tl
 {\exp_args:Ne \tl_head:n { \cs_to_str:N #1 } }
\str_if_eq:VnTF \l__fun_variable_type_tl { g } { \prg_return_true: }
{ \str_if_eq:VnTF \l__fun_variable_type_tl { c } { \prg_return_true: }
{ \prg_return_false: }
}
}

%% We must not put an assignment inside a group
\cs_new_protected:Npn \__fun_do_assignment:Nnn #1 #2 #3
{ \__fun_group_end:
 \__fun_if_global_variable:NTF #1 { #2 } { #3 }
 \__fun_group_begin:
}

\bool_new:N \l__fun_scoping_bool
\cs_new_protected:Npn \__fun_scoping_true:
{ \cs_set_eq:NN \__fun_group_begin: \group_begin:
 \cs_set_eq:NN \__fun_group_end: \group_end:
}
\cs_new_protected:Npn \__fun_scoping_false:
{ \cs_set_eq:NN \__fun_group_begin: \scan_stop:
 \cs_set_eq:NN \__fun_group_end: \scan_stop:
}
\cs_new_protected:Npn \__fun_scoping_set:
{ \bool_if:NTF \l__fun_scoping_bool { \__fun_scoping_true: } { \__fun_scoping_false: }
}

\bool_new:N \l__fun_tracing_bool
\tl_new:N \l__tracing_text_tl
\cs_new_protected:Npn \__fun_tracing_log_on:n #1
{ \tl_set:Ne \l__tracing_text_tl
 { \prg_replicate:nn { \int_eval:n { (\g__fun_nesting_level_int - 1) * 4 } } { - } }
 \tl_put_right:Nn \l__tracing_text_tl { #1 }
 \iow_log:V \l__tracing_text_tl
}
\cs_generate_variant:Nn \__fun_tracing_log_on:n { e, V }
\cs_new_protected:Npn \__fun_tracing_log_off:n #1 { }

\cs_new_protected:Npn \__fun_tracing_log_on:n #1 { }
\textbf{CHAPTER 11. THE SOURCE CODE}

\cs_generate_variant:Nn \_fun_tracing_log_off:n \{ e, V \}
\cs_new_protected:Npn \_fun_tracing_true:
{ \cs_set_eq:NN \_fun_tracing_log:n \_fun_tracing_log_on:n
  \cs_set_eq:NN \_fun_tracing_log:e \_fun_tracing_log_on:e
  \cs_set_eq:NN \_fun_tracing_log:V \_fun_tracing_log_on:V
}
\cs_new_protected:Npn \_fun_tracing_false:
{ \cs_set_eq:NN \_fun_tracing_log:n \_fun_tracing_log_off:n
  \cs_set_eq:NN \_fun_tracing_log:e \_fun_tracing_log_off:e
  \cs_set_eq:NN \_fun_tracing_log:V \_fun_tracing_log_off:V
}
\cs_new_protected:Npn \_fun_tracing_set:
{ \bool_if:NTF \l__fun_tracing_bool
  { \_fun_tracing_true: } { \_fun_tracing_false: }
}
\keys_define:nn { functional }
{ scoping .bool_set:N = \l__fun_scoping_bool,
  tracing .bool_set:N = \l__fun_tracing_bool,
}
\NewDocumentCommand \Functional { m }
{ \keys_set:nn { functional } { #1 }
  \_fun_scoping_set:
  \_fun_tracing_set:
}
\Functional { scoping = false, tracing = false }
\cs_new_protected:Npn \_fun_ignore_spaces_on:
{ \ExplSyntaxOn
  \char_set_catcode_math_subscript:N \_
  \char_set_catcode_other:N : 
\}
\cs_set_eq:NN \IgnoreSpacesOn \_fun_ignore_spaces_on:
\cs_set_eq:NN \IgnoreSpacesOff \ExplSyntaxOff

\section{Interfaces for Argument Using (Use)}
\PrgNewFunction \Name { m }
{ \exp_args:Nc \_fun_put_result:n \{ #1 \}
}
\cs_set_eq:NN \UseName \Name
\PrgNewFunction \Value { M }
11.3 Interfaces for Control Structures (Bool)

\bool_const:Nn \cTrueBool { \c_true_bool }
\bool_const:Nn \cFalseBool { \c_false_bool }

\bool_new:N \lTmpaBool \bool_new:N \lTmpbBool \bool_new:N \lTmpcBool
\bool_new:N \lTmpiBool \bool_new:N \lTmpjBool \bool_new:N \lTmpkBool
\bool_new:N \l@Funx@Bool \bool_new:N \l@Funy@Bool \bool_new:N \l@Funz@Bool
\bool_new:N \gTmpaBool \bool_new:N \gTmpbBool \bool_new:N \gTmpcBool
\bool_new:N \gTmpiBool \bool_new:N \gTmpjBool \bool_new:N \gTmpkBool
\bool_new:N \g@Funx@Bool \bool_new:N \g@Funy@Bool \bool_new:N \g@Funz@Bool

\PrgNewFunction \BoolNew { M } { \bool_new:N #1 }
\PrgNewFunction \BoolConst { M m } { \bool_const:Nn #1 { #2 } }
\PrgNewFunction \BoolSet { M m } {
  \__fun_do_assignment:Nnn #1 { \bool_gset:Nn #1 { #2 } } { \bool_set:Nn #1 { #2 } }
}
\PrgNewFunction \BoolSetTrue { M }
  \__fun_do_assignment:Nnn #1 { \bool_gset_true:N #1 } { \bool_set_true:N #1 }
\PrgNewFunction \BoolSetFalse { M }
  \__fun_do_assignment:Nnn #1 { \bool_gset_false:N #1 } { \bool_set_false:N #1 }
\PrgNewFunction \BoolSetEq { M M }
{ \_fun_do_assignment:Nnn \texttt{#1} 
  \{ \bool_gset_eq:NN \texttt{#1} \texttt{#2} \} \{ \bool_set_eq:NN \texttt{#1} \texttt{#2} \}
}

\PrgNewFunction \BoolLog { m } \{ \bool_log:n \texttt{#1} \}

\PrgNewFunction \BoolVarLog { M } \{ \bool_log:N \texttt{#1} \}

\PrgNewFunction \BoolShow { m } \{ \bool_show:n \texttt{#1} \}

\PrgNewFunction \BoolVarShow { M } \{ \bool_show:N \texttt{#1} \}

\PrgNewConditional \BoolIfExist { M }
{ \bool_if_exist:NTF \texttt{#1} \{ \Result \{ \cTrueBool \} \} \{ \Result \{ \cFalseBool \} \}
}

\PrgNewConditional \BoolVarIf { M } \{ \Result \texttt{#1} \}

\PrgNewConditional \BoolVarNot { M }
{ \bool_if:NTF \texttt{#1} 
  \{ \Result \{ \cFalseBool \} \} \{ \Result \{ \cTrueBool \} \}
}

\PrgNewConditional \BoolVarAnd { M M }
{ \bool_lazy_and:nnTF \texttt{#1} \texttt{#2} 
  \{ \Result \{ \cTrueBool \} \} \{ \Result \{ \cFalseBool \} \}
}

\PrgNewConditional \BoolVarOr { M M }
{ \bool_lazy_or:nnTF \texttt{#1} \texttt{#2} 
  \{ \Result \{ \cTrueBool \} \} \{ \Result \{ \cFalseBool \} \}
}

\PrgNewConditional \BoolVarXor { M M }
{ \bool_xor:nnTF \texttt{#1} \texttt{#2} 
  \{ \Result \{ \cTrueBool \} \} \{ \Result \{ \cFalseBool \} \}
}

11.4 Interfaces for Token Lists (Tl)

\tl_set_eq:NN \cEmptyTl \c_empty_tl
\tl_set_eq:NN \cSpaceTl \c_space_tl
\tl_set_eq:NN \cNoValueTl \c_novalue_tl
\tl_new:N \lTmpaTl \tl_new:N \lTmpbTl \tl_new:N \lTmpcTl
\tl_new:N \lTmpdTl \tl_new:N \lTmpdTl \tl_new:N \lTmpkTl
\tl_new:N \l@FunxTl \tl_new:N \l@FunyTl \tl_new:N \l@FunzTl
\tl_new:N \g_Tmpa_Tl \tl_new:N \g_Tmpb_Tl \tl_new:N \g_Tmpc_Tl
\tl_new:N \g_Tmpd_Tl \tl_new:N \g_Tmpj_Tl \tl_new:N \g_Tmpk_Tl
\tl_new:N \g_Funx_Tl \tl_new:N \g_Funy_Tl \tl_new:N \g_Funz_Tl

\PrgNewFunction \TlNew { M } { \tl_new:N #1 }
\PrgNewFunction \TlLog { m } { \tl_log:n { #1 } }
\PrgNewFunction \TlVarLog { M } { \tl_log:N #1 }
\PrgNewFunction \TlShow { m } { \tl_show:n { #1 } }
\PrgNewFunction \TlVarShow { M } { \tl_show:N #1 }
\PrgNewFunction \TlUse { M } { \Result { \Value #1 } }
\PrgNewFunction \TlToStr { m } { \Expand { \tl_to_str:n { #1 } } }
\PrgNewFunction \TlVarToStr { M } { \Expand { \tl_to_str:N #1 } }
\PrgNewFunction \TlConst { M m } { \tl_const:Nn #1 { #2 } }
\PrgNewFunction \TlSet { M m }
\{ \__fun_do_assignment:Nnn #1 { \tl_gset:Nn #1 {#2} } { \tl_set:Nn #1 {#2} } \}
\PrgNewFunction \TlSetEq { M M }
\{ \__fun_do_assignment:Nnn #1 { \tl_gset_eq:NN #1 #2 } { \tl_set_eq:NN #1 #2 } \}
\PrgNewFunction \TlConcat { M M M }
\{ \__fun_do_assignment:Nnn #1 { \tl_gconcat:NNN #1 #2 #3 } { \tl_concat:NNN #1 #2 #3 } \}
\PrgNewFunction \TlClear { M }
\{ \__fun_do_assignment:Nnn #1 { \tl_gclear:N #1 } { \tl_clear:N #1 } \}
\PrgNewFunction \TlClearNew { M }
\{ \__fun_do_assignment:Nnn #1 { \tl_gclear_new:N #1 } { \tl_clear_new:N #1 } \}
\PrgNewFunction \TlPutLeft { M m }
\{ \__fun_do_assignment:Nnn #1 { \tl_gput_left:Nn #1 {#2} } { \tl_put_left:Nn #1 {#2} } \}
CHAPTER 11. THE SOURCE CODE

\PrgNewFunction \TlPutRight { M m }
\{ \_\_fun_do_assignment:Nnn #1
\{ \tl_gput_right:Nn #1 {#2} \} \{ \tl_put_right:Nn #1 {#2} \}
\}

\PrgNewFunction \TlReplaceOnce { M m m }
\{ \_\_fun_do_assignment:Nnn #1
\{ \tl_greplace_once:Nnn #1 {#2} {#3} \} \{ \tl_replace_once:Nnn #1 {#2} {#3} \}
\}

\PrgNewFunction \TlReplaceAll { M m m }
\{ \_\_fun_do_assignment:Nnn #1
\{ \tl_greplace_all:Nnn #1 {#2} {#3} \} \{ \tl_replace_all:Nnn #1 {#2} {#3} \}
\}

\PrgNewFunction \TlRemoveOnce { M m }
\{ \_\_fun_do_assignment:Nnn #1
\{ \tl_gremove_once:Nn #1 {#2} \} \{ \tl_remove_once:Nn #1 {#2} \}
\}

\PrgNewFunction \TlRemoveAll { M m }
\{ \_\_fun_do_assignment:Nnn #1
\{ \tl_gremove_all:Nn #1 {#2} \} \{ \tl_remove_all:Nn #1 {#2} \}
\}

\PrgNewFunction \TlTrimSpaces { m } \{ \Expand { \tl_trim_spaces:n { #1 } } \}

\PrgNewFunction \TlVarTrimSpaces { M }
\{ \_\_fun_do_assignment:Nnn #1 \{ \tl_gtrim_spaces:N #1 \} \{ \tl_trim_spaces:N #1 \}
\}

\PrgNewFunction \TlCount { m } \{ \Expand { \tl_count:n { #1 } } \}

\PrgNewFunction \TlVarCount { M } \{ \Expand { \tl_count:N #1 } \}

\PrgNewFunction \TlHead { m } \{ \Expand { \tl_head:n { #1 } } \}

\PrgNewFunction \TlVarHead { M } \{ \Expand { \tl_head:N #1 } \}

\PrgNewFunction \TlTail { m } \{ \Expand { \tl_tail:n { #1 } } \}

\PrgNewFunction \TlVarTail { M } \{ \Expand { \tl_tail:N #1 } \}

\PrgNewFunction \TlItem { m m } \{ \Expand { \tl_item:nn {#1} {#2} } \}

\PrgNewFunction \TlVarItem { M m } \{ \Expand { \tl_item:Nn #1 {#2} } \}

\PrgNewFunction \TlRandItem { m } \{ \Expand { \tl_rand_item:n {#1} } \}
\PrfNewFunction \TlVarRandItem { M } \{ \Expand \{ \tl_rand_item:N #1 \} \}

\PrfNewFunction \TlVarCase { M m } \{ \tl_case:Nn \{ #1 \} \{ #2 \} \}
\PrfNewFunction \TlVarCaseT { M m n } \{ \tl_case:NnT \{ #1 \} \{ #2 \} \{ #3 \} \}
\PrfNewFunction \TlVarCaseF { M m n } \{ \tl_case:NnF \{ #1 \} \{ #2 \} \{ #3 \} \}
\PrfNewFunction \TlVarCaseTF { M m n n } \{ \tl_case:NnTF \{ #1 \} \{ #2 \} \{ #3 \} \{ #4 \} \}

\PrfNewFunction \TlMapInline { m n } \{
\tl_map_inline:nn \{ #1 \} \{ #2 \}
\}

\PrfNewFunction \TlVarMapInline { M n } \{
\tl_map_inline:Nn #1 \{ #2 \}
\}

\PrfNewFunction \TlMapVariable { m M n } \{
\tl_map_variable:nNn \{ #1 \} \{ #2 \} \{ #3 \}
\}

\PrfNewFunction \TlVarMapVariable { M M n } \{
\tl_map_variable:NNn #1 \{ #2 \} \{ #3 \}
\}

\PrfNewConditional \TlIfExist { M } \{
\tl_if_exist:NTF \{ #1 \} \{ \Result \{ \cTrueBool \} \} \{ \Result \{ \cFalseBool \} \}
\}

\PrfNewConditional \TlIfEmpty { m } \{
\tl_if_empty:nTF \{ #1 \} \{ \Result \{ \cTrueBool \} \} \{ \Result \{ \cFalseBool \} \}
\}

\PrfNewConditional \TlVarIfEmpty { M } \{
\tl_if_empty:NTF \{ #1 \} \{ \Result \{ \cTrueBool \} \} \{ \Result \{ \cFalseBool \} \}
\}

\PrfNewConditional \TlIfBlank { m } \{
\tl_if_blank:nTF \{ #1 \} \{ \Result \{ \cTrueBool \} \} \{ \Result \{ \cFalseBool \} \}
\}

\PrfNewConditional \TlIfEq { m m } \{
\tl_if_eq:nnTF \{ #1 \} \{ #2 \} \{ \Result \{ \cTrueBool \} \} \{ \Result \{ \cFalseBool \} \}
\}

\PrfNewConditional \TlVarIfEq { M M } \{
\tl_if_eq:NNTF \{ #1 \} \{ #2 \} \{ \Result \{ \cTrueBool \} \} \{ \Result \{ \cFalseBool \} \}
\}
CHAPTER 11. THE SOURCE CODE

\PrgNewConditional \TlIfIn { m m }
\{ \tl_if_in:nnTF {#1} {#2} { \Result { \cTrueBool } } { \Result { \cFalseBool } } \}
\}

\PrgNewConditional \TlVarIfIn { M m }
\{ \tl_if_in:NnTF #1 {#2} { \Result { \cTrueBool } } { \Result { \cFalseBool } } \}
\}

\PrgNewConditional \TlIfSingle { m }
\{ \tl_if_single:nTF {#1} { \Result { \cTrueBool } } { \Result { \cFalseBool } } \}
\}

\PrgNewConditional \TlVarIfSingle { M }
\{ \tl_if_single:NTF #1 { \Result { \cTrueBool } } { \Result { \cFalseBool } } \}
\}

11.5 Interfaces for Strings (Str)

\str_set_eq:NN \cAmpersandStr \c_ampersand_str
\str_set_eq:NN \cAttignStr     \c_atsign_str
\str_set_eq:NN \cBackslashStr  \c_backslash_str
\str_set_eq:NN \cLeftBraceStr  \c_left_brace_str
\str_set_eq:NN \cRightBraceStr \c_right_brace_str
\str_set_eq:NN \cCircumflexStr \c_circumflex_str
\str_set_eq:NN \cColonStr      \c_colon_str
\str_set_eq:NN \cDollarStr     \c_dollar_str
\str_set_eq:NN \cHashStr       \c_hash_str
\str_set_eq:NN \cPercentStr    \c_percent_str
\str_set_eq:NN \cTildeStr      \c_tilde_str
\str_set_eq:NN \cUnderscoreStr \
c_underscore_str
\str_set_eq:NN \cZeroStr       \c_zero_str

\str_new:N \lTmpaStr   \str_new:N \lTmpbStr   \str_new:N \lTmpcStr
\str_new:N \lTmpiStr   \str_new:N \lTmpjStr   \str_new:N \lTmpkStr
\str_new:N \l@Funx@Str \str_new:N \l@Funy@Str \str_new:N \l@Funz@Str
\str_new:N \gTmpaStr   \str_new:N \gTmpbStr   \str_new:N \gTmpcStr
\str_new:N \gTmpiStr   \str_new:N \gTmpjStr   \str_new:N \gTmpkStr
\str_new:N \g@Funx@Str \str_new:N \g@Funy@Str \str_new:N \g@Funz@Str

\PrgNewFunction \StrNew { M } { \str_new:N #1 }
\PrgNewFunction \StrLog { m } { \str_log:n { #1 } }
\PrgNewFunction \StrVarLog { M } { \str_log:N #1 }
\PrgNewFunction \StrShow { m } { \str_show:n { #1 } }
\PrgNewFunction \StrVarShow { M } { \str_show:N #1 }

\PrgNewFunction \StrUse { M } { \Result { \Value #1 } }

\PrgNewFunction \StrConst { M m } { \str_const:Nn #1 {#2} }

\PrgNewFunction \StrSet { M m }
{ \__fun_do_assignment:Nnn #1 { \str_gset:Nn #1 {#2} } { \str_set:Nn #1 {#2} }
}

\PrgNewFunction \StrSetEq { M M }
{ \__fun_do_assignment:Nnn #1 { \str_gset_eq:NN #1 #2 #2 } { \str_set_eq:NN #1 #2 #2 }
}

\PrgNewFunction \StrConcat { M M M }
{ \__fun_do_assignment:Nnn #1 { \str_gconcat:NNN #1 #2 #3 } { \str_concat:NNN #1 #2 #3 }
}

\PrgNewFunction \StrClear { M }
{ \__fun_do_assignment:Nnn #1 { \str_gclear:N #1 } { \str_clear:N #1 }
}

\PrgNewFunction \StrClearNew { M }
{ \__fun_do_assignment:Nnn #1 { \str_gclear_new:N #1 } { \str_clear_new:N #1 }
}

\PrgNewFunction \StrPutLeft { M m }
{ \__fun_do_assignment:Nnn #1 { \str_gput_left:Nn #1 {#2} } { \str_put_left:Nn #1 {#2} }
}

\PrgNewFunction \StrPutRight { M m }
{ \__fun_do_assignment:Nnn #1 { \str_gput_right:Nn #1 {#2} } { \str_put_right:Nn #1 {#2} }
}

\PrgNewFunction \StrReplaceOnce { M m m }
{ \__fun_do_assignment:Nnn #1 { \str_greplace_once:Nnn #1 {#2} {#3} } { \str_replace_once:Nnn #1 {#2} {#3} }
}

\PrgNewFunction \StrReplaceAll { M m m }
{ \__fun_do_assignment:Nnn #1 { \str_greplace_all:Nnn #1 {#2} {#3} } { \str_replace_all:Nnn #1 {#2} {#3} }
}
CHAPTER 11. THE SOURCE CODE

\PrgNewFunction \StrRemoveOnce { M m }
{ \_fun_do_assignment:Nnn #1
  { \str_gremove_once:Nn #1 (#2) } { \str_remove_once:Nn #1 (#2) }
}

\PrgNewFunction \StrRemoveAll { M m }
{ \_fun_do_assignment:Nnn #1
  { \str_gremove_all:Nn #1 (#2) } { \str_remove_all:Nn #1 (#2) }
}

%% Avoid naming conflict with xstring package
\cs_if_exist:NF \StrCount
  { \PrgNewFunction \StrCount { m } { \Expand { \str_count:n { #1 } } } }

%% Provide another name for \StrCount function
\PrgNewFunction \StrSize { m } { \Expand { \str_count:n { #1 } } }

\PrgNewFunction \StrVarCount { M } { \Expand { \str_count:N #1 } }

\PrgNewFunction \StrHead { m } { \Expand { \str_head:n { #1 } } }

\PrgNewFunction \StrVarHead { M } { \Expand { \str_head:N #1 } }

\PrgNewFunction \StrTail { m } { \Expand { \str_tail:n { #1 } } }

\PrgNewFunction \StrVarTail { M } { \Expand { \str_tail:N #1 } }

\PrgNewFunction \StrItem { m m } { \Expand { \str_item:nn {#1} {#2} } }

\PrgNewFunction \StrVarItem { M m } { \Expand { \str_item:Nn #1 {#2} } }

\PrgNewFunction \StrCase   { m m }     { \str_case:nn {#1} {#2} }
\PrgNewFunction \StrCaseT  { m m n }   { \str_case:nnT {#1} {#2} {#3} }
\PrgNewFunction \StrCaseF  { m m n }   { \str_case:nnF {#1} {#2} {#3} }
\PrgNewFunction \StrCaseTF { m m n n } { \str_case:nnTF {#1} {#2} {#3} {#4} }

\PrgNewFunction \StrMapInline { m n }
{ \str_map_inline:nn {#1} {#2} }

\PrgNewFunction \StrVarMapInline { M n }
{ \str_map_inline:Nn #1 {#2} }

\PrgNewFunction \StrMapVariable { m M n }
{ \str_map_variable:nNn {#1} {#2} {#3} }

\PrgNewFunction \StrVarMapVariable { M M n }
\{ \texttt{str\_map\_variable:NNn} \#1 \#2 \{#3\} \}

\texttt{PrgNewConditional \ StrIfExist \{ M \}}
\{
\texttt{str\_if\_exist:NTF} \#1 \{ \texttt{Result} \{ \texttt{cTrueBool} \} \} \{ \texttt{Result} \{ \texttt{cFalseBool} \} \}
\}

\texttt{PrgNewConditional \ StrVarIfEmpty \{ M \}}
\{
\texttt{str\_if\_empty:NTF} \#1 \{ \texttt{Result} \{ \texttt{cTrueBool} \} \} \{ \texttt{Result} \{ \texttt{cFalseBool} \} \}
\}

\texttt{PrgNewConditional \ StrIfEq \{ m \ m \}}
\{
\texttt{str\_if\_eq:nnTF} \#1 \#2 \{ \texttt{Result} \{ \texttt{cTrueBool} \} \} \{ \texttt{Result} \{ \texttt{cFalseBool} \} \}
\}

\texttt{PrgNewConditional \ StrVarIfEq \{ M \ M \}}
\{
\texttt{str\_if\_eq:NNTF} \#1 \#2 \{ \texttt{Result} \{ \texttt{cTrueBool} \} \} \{ \texttt{Result} \{ \texttt{cFalseBool} \} \}
\}

\texttt{PrgNewConditional \ StrIfIn \{ m \ m \}}
\{
\texttt{str\_if\_in:nnTF} \#1 \#2 \{ \texttt{Result} \{ \texttt{cTrueBool} \} \} \{ \texttt{Result} \{ \texttt{cFalseBool} \} \}
\}

\texttt{PrgNewConditional \ StrVarIfIn \{ M \ m \}}
\{
\texttt{str\_if\_in:NnTF} \#1 \#2 \{ \texttt{Result} \{ \texttt{cTrueBool} \} \} \{ \texttt{Result} \{ \texttt{cFalseBool} \} \}
\}

%% Avoid naming conflict with xstring package
\texttt{cs\_if\_exist:NF \ StrCompare}
\{
\texttt{PrgNewConditional \ StrCompare \{ m N m \}}
\{
\texttt{str\_compare:nNnTF} \#1 \#2 \{#3\}
\{ \texttt{Result} \{ \texttt{cTrueBool} \} \}
\{ \texttt{Result} \{ \texttt{cFalseBool} \} \}
\}
\%

%% Provide another name for \StrCompare function
\texttt{PrgNewConditional \ SStrIfCompare \{ m N m \}}
\{
\texttt{str\_compare:nNnTF} \#1 \#2 \{#3\}
\{ \texttt{Result} \{ \texttt{cTrueBool} \} \}
\{ \texttt{Result} \{ \texttt{cFalseBool} \} \}
\}

11.6 Interfaces for Integers (Int)
\texttt{cs\_set\_eq:NN \ cZeroInt \ c\_zero\_int}
\cs_set_eq:NN \cOneInt \c_one_int
\cs_set_eq:NN \cMaxInt \c_max_int
\cs_set_eq:NN \cMaxRegisterInt \c_max_register_int
\cs_set_eq:NN \cMaxCharInt \c_max_char_in
\int_new:N \lTmpaInt \int_new:N \lTmpbInt \int_new:N \lTmpcInt
\int_new:N \lTmpjInt \int_new:N \lTmpkInt
\int_new:N \l@Funx@Int \int_new:N \l@Funy@Int \int_new:N \l@Funz@Int
\int_new:N \gTmpaInt \int_new:N \gTmpbInt \int_new:N \gTmpcInt
\int_new:N \gTmpjInt \int_new:N \gTmpkInt
\int_new:N \g@Funx@Int \int_new:N \g@Funy@Int \int_new:N \g@Funz@Int

\PrgNewFunction \IntEval { m }
{
  \Result { \Expand { \int_eval:n { #1 } } }
}

\PrgNewFunction \IntMathAdd { m m }
{
  \int_set:Nn \l@Funx@Int { \int_eval:n { (#1) + (#2) } }
  \Result { \Value \l@Funx@Int }
}

\PrgNewFunction \IntMathSub { m m }
{
  \int_set:Nn \l@Funx@Int { \int_eval:n { (#1) - (#2) } }
  \Result { \Value \l@Funx@Int }
}

\PrgNewFunction \IntMathMult { m m }
{
  \int_set:Nn \l@Funx@Int { \int_eval:n { (#1) * (#2) } }
  \Result { \Value \l@Funx@Int }
}

\PrgNewFunction \IntMathDiv { m m }
{
  \Expand { \int_div_round:nn { #1 } { #2 } }
}

\PrgNewFunction \IntMathDivTruncate { m m }
{
  \Expand { \int_div_truncate:nn { #1 } { #2 } }
}

\PrgNewFunction \IntMathSign { m } { \Expand { \int_sign:n { #1 } } }
\PrgNewFunction \IntMathAbs { m } { \Expand { \int_abs:n { #1 } } }
\PrgNewFunction \IntMathMax { m m } { \Expand { \int_max:nn { #1 } { #2 } } }
\PrgNewFunction \IntMathMin { m m } { \Expand { \int_min:nn { #1 } { #2 } } }
\PrgNewFunction \IntMathMod { m m } { \Expand { \int_mod:nn { #1 } { #2 } } }
\PrgNewFunction \IntMathRand { m m } { \Expand { \int_rand:nn { #1 } { #2 } } }

\PrgNewFunction \IntNew { M } { \int_new:N #1 }

\PrgNewFunction \IntConst { M m } { \int_const:Nn #1 { #2 } }

\PrgNewFunction \IntLog { m } { \int_log:n { #1 } }

\PrgNewFunction \IntVarLog { M } { \int_log:N #1 }

\PrgNewFunction \IntShow { m } { \int_show:n { #1 } }

\PrgNewFunction \IntVarShow { M } { \int_show:N #1 }

\PrgNewFunction \IntUse { M } { \Result { \Value #1 } }

\PrgNewFunction \IntSet { M m }
{ \__fun_do_assignment:Nnn #1 { \int_gset:Nn #1 {#2} } { \int_set:Nn #1 {#2} } }

\PrgNewFunction \IntZero { M }
{ \__fun_do_assignment:Nnn #1 { \int_gzero:N #1 } { \int_zero:N #1 } }

\PrgNewFunction \IntZeroNew { M }
{ \__fun_do_assignment:Nnn #1 { \int_gzero_new:N #1 } { \int_zero_new:N #1 } }

\PrgNewFunction \IntSetEq { M M }
{ \__fun_do_assignment:Nnn #1 { \int_gset_eq:NN #1 #2 } { \int_set_eq:NN #1 #2 } }

\PrgNewFunction \IntIncr { M }
{ \__fun_do_assignment:Nnn #1 { \int_gincr:N #1 } { \int_incr:N #1 } }

\PrgNewFunction \IntDecr { M }
{ \__fun_do_assignment:Nnn #1 { \int_gdecr:N #1 } { \int_decr:N #1 } }

\PrgNewFunction \IntAdd { M m }
{ \__fun_do_assignment:Nnn #1 { \int_gadd:Nn #1 {#2} } { \int_add:Nn #1 {#2} } }

\PrgNewFunction \IntSub { M m }
{ \__fun_do_assignment:Nnn #1 { \int_gsub:Nn #1 {#2} } { \int_sub:Nn #1 {#2} } }
11.7 Interfaces for Floating Point Numbers (Fp)

\fp_set_eq:NN \cZeroFp \c_zero_fp
\fp_set_eq:NN \cMinusZeroFp \c_minus_zero_fp
\fp_set_eq:NN \cOneFp \c_one_fp
\fp_set_eq:NN \cInfFp \c_inf_fp
\fp_set_eq:NN \cMinusInfFp \c_minus_inf_fp
\fp_set_eq:NN \cEFp \c_e_fp
\fp_set_eq:NN \cPiFp \c_pi_fp
\fp_set_eq:NN \cOneDegreeFp \c_one_degree_fp

\fp_new:N \lTmpaFp \fp_new:N \lTmpbFp \fp_new:N \lTmpcFp
\fp_new:N \l@Funx@Fp \fp_new:N \l@Funy@Fp \fp_new:N \l@Funz@Fp
\fp_new:N \gTmpaFp \fp_new:N \gTmpbFp \fp_new:N \gTmpcFp
\PrgNewFunction \FpEval { m }
{ \Result { \Expand { \fp_eval:n { \#1 } } } }

\PrgNewFunction \FpMathAdd { m m }
{ \fp_set:Nn \l@Funx@Fp { \fp_eval:n { (#1) + (#2) } } \Result { \FpUse \l@Funx@Fp } }

\PrgNewFunction \FpMathSub { m m }
{ \fp_set:Nn \l@Funx@Fp { \fp_eval:n { (#1) - (#2) } } \Result { \FpUse \l@Funx@Fp } }

\PrgNewFunction \FpMathMult { m m }
{ \fp_set:Nn \l@Funx@Fp { \fp_eval:n { (#1) * (#2) } } \Result { \FpUse \l@Funx@Fp } }

\PrgNewFunction \FpMathDiv { m m }
{ \fp_set:Nn \l@Funx@Fp { \fp_eval:n { (#1) / (#2) } } \Result { \FpUse \l@Funx@Fp } }

\PrgNewFunction \FpMathSign { m }
{ \Result { \Expand { \fp_sign:n { \#1 } } } }

\PrgNewFunction \FpMathAbs { m }
{ \Result { \Expand { \fp_abs:n { \#1 } } } }

\PrgNewFunction \FpMathMax { m m }
{ \Result { \Expand { \fp_max:nn { \#1 } { \#2 } } } }

\PrgNewFunction \FpMathMin { m m }
{ \Result { \Expand { \fp_min:nn { \#1 } { \#2 } } } }

\PrgNewFunction \FpNew { M } { \fp_new:N \#1 }
\PrgNewFunction \FpConst { M m } { \fp_const:Nn \#1 \{\#2\} }
\PrgNewFunction \FpUse { M } { \Result { \Expand { \fp_use:N #1 } } } \\
\PrgNewFunction \FpLog { m } { \fp_log:n { #1 } } \\
\PrgNewFunction \FpVarLog { M } { \fp_log:N #1 } \\
\PrgNewFunction \FpShow { m } { \fp_show:n { #1 } } \\
\PrgNewFunction \FpVarShow { M } { \fp_show:N #1 } \\
\PrgNewFunction \FpSet { M m } \\
{ \__fun_do_assignment:Nnn #1 { \fp_gset:Nn #1 {#2} } { \fp_set:Nn #1 {#2} } } \\
\PrgNewFunction \FpSetEq { M M } \\
{ \__fun_do_assignment:Nnn #1 { \fp_gset_eq:NN #1 #2 } { \fp_set_eq:NN #1 #2 } } \\
\PrgNewFunction \FpZero { M } \\
{ \__fun_do_assignment:Nnn #1 { \fp_gzero:N #1 } { \fp_zero:N #1 } } \\
\PrgNewFunction \FpZeroNew { M } \\
{ \__fun_do_assignment:Nnn #1 { \fp_gzero_new:N #1 } { \fp_zero_new:N #1 } } \\
\PrgNewFunction \FpAdd { M m } \\
{ \__fun_do_assignment:Nnn #1 { \fp_gadd:Nn #1 {#2} } { \fp_add:Nn #1 {#2} } } \\
\PrgNewFunction \FpSub { M m } \\
{ \__fun_do_assignment:Nnn #1 { \fp_gsub:Nn #1 {#2} } { \fp_sub:Nn #1 {#2} } } \\
\PrgNewFunction \FpStepInline { m m m n } \\
{ \fp_step_inline:nnnn { #1 } { #2 } { #3 } { #4 } } \\
\PrgNewFunction \FpStepVariable { m m M n } \\
{ \fp_step_variable:nnnNn { #1 } { #2 } { #3 } #4 { #5 } } \\
\PrgNewConditional \FpIfExist { M } \\
{ \fp_if_exist:NTF #1 { \Result { \cTrueBool } } { \Result { \cFalseBool } } }
\textbf{11.8 Interfaces for Dimensions (Dim)}

\begin{verbatim}
\cs_set_eq:NN \cMaxDim \c_max_dim
\cs_set_eq:NN \cZeroDim \c_zero_dim
\dim_new:N \lTmpaDim \dim_new:N \lTmpbDim \dim_new:N \lTmpcDim
\dim_new:N \lTmpiDim \dim_new:N \lTmpjDim \dim_new:N \lTmpkDim
\dim_new:N \l@Funx@Dim \dim_new:N \l@Funy@Dim \dim_new:N \l@Funz@Dim
\dim_new:N \gTmpaDim \dim_new:N \gTmpbDim \dim_new:N \gTmpcDim
\dim_new:N \gTmpiDim \dim_new:N \gTmpjDim \dim_new:N \gTmpkDim
\dim_new:N \g@Funx@Dim \dim_new:N \g@Funy@Dim \dim_new:N \g@Funz@Dim

\PrgNewFunction \DimEval { m }
\Result { \Expand { \dim_eval:n { #1 } } }

\PrgNewFunction \DimMathAdd { m m }
\Result { \Value \l@Funx@Dim }

\PrgNewFunction \DimMathSub { m m }
\Result { \Value \l@Funx@Dim }

\PrgNewFunction \DimMathSign { m }
\Result { \Expand { \dim_sign:n { #1 } } }

\PrgNewFunction \DimMathAbs { m }
\Result { \Expand { \dim_abs:n { #1 } } }

\PrgNewFunction \DimMathMax { m m }
\Result { \Expand { \dim_max:nn { #1 } { #2 } } }

\PrgNewFunction \DimMathMin { m m }
\Result { \Expand { \dim_min:nn { #1 } { #2 } } }
\end{verbatim}
\Result { \Expand { \dim_min:nn { #1 } { #2 } } }

\PrgNewFunction \DimMathRatio { m m }
{ \Result { \Expand { \dim_ratio:nn { #1 } { #2 } } }
}

\PrgNewFunction \DimNew { M } { \dim_new:N #1 }

\PrgNewFunction \DimConst { M m } { \dim_const:Nn #1 {#2} }

\PrgNewFunction \DimUse { M } { \Result { \Value #1 } }

\PrgNewFunction \DimLog { m } { \dim_log:n { #1 } }

\PrgNewFunction \DimVarLog { M } { \dim_log:N #1 }

\PrgNewFunction \DimShow { m } { \dim_show:n { #1 } }

\PrgNewFunction \DimVarShow { M } { \dim_show:N #1 }

\PrgNewFunction \DimSet { M m }
{ \_fun_do_assignment:Nnn #1 { \dim_gset:Nn #1 {#2} } { \dim_set:Nn #1 {#2} }
}

\PrgNewFunction \DimSetEq { M M }
{ \_fun_do_assignment:Nnn #1 { \dim_gset_eq:NN #1 #2 } { \dim_set_eq:NN #1 #2 }
}

\PrgNewFunction \DimZero { M }
{ \_fun_do_assignment:Nnn #1 { \dim_gzero:N #1 } { \dim_zero:N #1 }
}

\PrgNewFunction \DimZeroNew { M }
{ \_fun_do_assignment:Nnn #1 { \dim_gzero_new:N #1 } { \dim_zero_new:N #1 }
}

\PrgNewFunction \DimAdd { M m }
{ \_fun_do_assignment:Nnn #1 { \dim_gadd:Nn #1 {#2} } { \dim_add:Nn #1 {#2} }
}

\PrgNewFunction \DimSub { M m }
{ \_fun_do_assignment:Nnn #1 { \dim_gsub:Nn #1 {#2} } { \dim_sub:Nn #1 {#2} }
}

\PrgNewFunction \DimStepInline { m m m n }
{
\textbf{CHAPTER 11. THE SOURCE CODE}

\begin{verbatim}
\dim_step_inline:nnnn { #1 } { #2 } { #3 } { #4 }

\PrgNewFunction \DimStepVariable { m m m M n }
{ \dim_step_variable:nnnn { #1 } { #2 } { #3 } { #4 } }

\PrgNewConditional \DimIfExist { M }
{ \dim_if_exist:NTF #1 { \Result { \cTrueBool } } { \Result { \cFalseBool } } }

\PrgNewConditional \DimCompare { m N m }
{ \dim_compare:nNnTF {#1} #2 {#3} { \Result { \cTrueBool } } { \Result { \cFalseBool } } }

\PrgNewFunction \DimCase   { m m }     { \dim_case:nn {#1} {#2} }
\PrgNewFunction \DimCaseT  { m m n }   { \dim_case:nnT {#1} {#2} {#3} }
\PrgNewFunction \DimCaseF  { m m n }   { \dim_case:nnF {#1} {#2} {#3} }
\PrgNewFunction \DimCaseTF { m m n n } { \dim_case:nnTF {#1} {#2} {#3} {#4} }

11.9 Interfaces for Sorting Functions (Sort)
\cs_set_eq:NN \SortReturnSame \sort_return_same:
\cs_set_eq:NN \SortReturnSwapped \sort_return_swapped:

11.10 Interfaces for Comma Separated Lists (Clist)
\clist_new:N \lTmpaClist \clist_new:N \lTmpbClist \clist_new:N \lTmpcClist
\clist_new:N \lTmpiClist \clist_new:N \lTmpjClist \clist_new:N \lTmpkClist
\clist_new:N \l@Funx@Clist \clist_new:N \l@Funy@Clist \clist_new:N \l@Funz@Clist
\clist_new:N \gTmpaClist \clist_new:N \gTmpbClist \clist_new:N \gTmpcClist
\clist_new:N \gTmpiClist \clist_new:N \gTmpjClist \clist_new:N \gTmpkClist
\clist_new:N \g@Funx@Clist \clist_new:N \g@Funy@Clist \clist_new:N \g@Funz@Clist
\clist_set_eq:NN \cEmptyClist \c_empty_clist

\PrgNewFunction \ClistNew { M } \clist_new:N #1
\PrgNewFunction \ClistLog { m } \clist_log:n #1
\PrgNewFunction \ClistVarLog { M } \clist_log:N #1
\PrgNewFunction \ClistShow { m } \clist_show:n #1
\PrgNewFunction \ClistVarShow { M } \clist_show:N #1
\PrgNewFunction \ClistVarJoin { M m }
\end{verbatim}
\{  
\Expand { \clist_use:Nn #1 { #2 }  }  
\}  

\PrgNewFunction \ClistVarJoinExtended { M m m m }  
{  
\Expand { \clist_use:Nnnn #1 { #2 } { #3 } { #4 }  }  
}  

\PrgNewFunction \ClistJoin { m m }  
{  
\Expand { \clist_use:nn { #1 } { #2 }  }  
}  

\PrgNewFunction \ClistJoinExtended { m m m m }  
{  
\Expand { \clist_use:nnnn { #1 } { #2 } { #3 } { #4 }  }  
}  

\PrgNewFunction \ClistConst { M m } { \clist_const:Nn #1 { #2 }  }  

\PrgNewFunction \ClistSet { M m }  
{  
\__fun_do_assignment:Nnn { \clist_gset:Nn #1 (#2) } { \clist_set:Nn #1 (#2) }  
}  

\PrgNewFunction \ClistSetEq { M M }  
{  
\__fun_do_assignment:Nnn { \clist_gset_eq:NN #1 #2 } { \clist_set_eq:NN #1 #2 }  
}  

\PrgNewFunction \ClistSetFromSeq { M M }  
{  
\__fun_do_assignment:Nnn { \clist_gset_from_seq:NN #1 #2 } { \clist_set_from_seq:NN #1 #2 }  
}  

\PrgNewFunction \ClistConcat { M M M }  
{  
\__fun_do_assignment:Nnn { \clist_gconcat:NNN #1 #2 #3 } { \clist_concat:NNN #1 #2 #3 }  
}  

\PrgNewFunction \ClistClear { M }  
{  
\__fun_do_assignment:Nnn { \clist_gclear:N #1 } { \clist_clear:N #1 }  
}  

\PrgNewFunction \ClistClearNew { M }  
{  
\__fun_do_assignment:Nnn { \clist_gclear_new:N #1 } { \clist_clear_new:N #1 }  
}  

\PrgNewFunction \ClistPutLeft { M m }
\__fun_do_assignment:Nnn #1
  \clist_gput_left:Nn #1 {#2} \clist_put_left:Nn #1 {#2}
\}

\PrgNewFunction \ClistPutRight \{ M m \}
  \__fun_do_assignment:Nnn #1
  \clist_gput_right:Nn #1 {#2} \clist_put_right:Nn #1 {#2}
\}

\PrgNewFunction \ClistRemoveDuplicates \{ M \}
  \__fun_do_assignment:Nnn #1
  \clist_gremove_duplicates:N #1 \clist_remove_duplicates:N #1
\}

\PrgNewFunction \ClistRemoveAll \{ M m \}
  \__fun_do_assignment:Nnn #1
  \clist_gremove_all:Nn #1 {#2} \clist_remove_all:Nn #1 {#2}
\}

\PrgNewFunction \ClistReverse \{ M \}
  \__fun_do_assignment:Nnn #1
  \clist_greverse:N \clist_reverse:N #1
\}

\PrgNewFunction \ClistSort \{ M m \}
  \__fun_do_assignment:Nnn \clist_gsort:Nn #1 {#2} \clist_sort:Nn #1 {#2}
\}

\PrgNewFunction \ClistCount \{ m \}
  \Result \{ \Expand \clist_count:n \{ #1 \} \}
\}

\PrgNewFunction \ClistVarCount \{ m \}
  \Result \{ \Expand \clist_count:N \{ #1 \} \}
\}

\PrgNewFunction \ClistGet \{ M M \} \{ \clist_get:NN \{ #1 \} \}
\PrgNewFunction \ClistGetT \{ M M n \} \{ \clist_get:NNT \{ #1 \} {#2} \}
\PrgNewFunction \ClistGetF \{ M M n \} \{ \clist_get:NNF \{ #1 \} {#2} \}
\PrgNewFunction \ClistGetTF \{ M M n \} \{ \clist_get:NNTF \{ #1 \} {#2} \}
\PrgNewFunction \ClistCount \{ m \}
  \Result \{ \Expand \clist_count:n \{ #1 \} \}
\}

\PrgNewFunction \ClistPop \{ M M \}
  \__fun_do_assignment:Nnn \clist_gpop:NN \{ #1 \} \clist_pop:NN \{ #1 \}
\}

\PrgNewFunction \ClistPopT \{ M M n \}
  \__fun_do_assignment:Nnn \clist_gpop:NN \{ #1 \} \clist_pop:NN \{ #1 \
\_\_fun\_do\_assignment:Nnn #1
{ \clist_gpop:NNT #1 #2 {#3} } { \clist_pop:NNT #1 #2 {#3} }
}
\PrgNewFunction \ClistPopF { M M n }
{ \_\_fun\_do\_assignment:Nnn #1
{ \clist_gpop:NNF #1 #2 {#3} } { \clist_pop:NNF #1 #2 {#3} }
}
\PrgNewFunction \ClistPopTF { M M n n }
{ \_\_fun\_do\_assignment:Nnn #1
{ \clist_gpop:NNTF #1 #2 {#3} {#4} } { \clist_pop:NNTF #1 #2 {#3} {#4} }
}
\PrgNewFunction \ClistPush { M m }
{ \_\_fun\_do\_assignment:Nnn #1
{ \clist_gpush:Nn #1 {#2} } { \clist_push:Nn #1 {#2} }
}
\PrgNewFunction \ClistItem { m m } { \Expand { \clist_item:nn {#1} {#2} } }
\PrgNewFunction \ClistVarItem { M m } { \Expand { \clist_item:Nn #1 {#2} } }
\PrgNewFunction \ClistRandItem { m } { \Expand { \clist_rand_item:n {#1} } }
\PrgNewFunction \ClistVarRandItem { M } { \Expand { \clist_rand_item:N #1 } }
\PrgNewFunction \ClistMapInline { m n }
{ \clist_map_inline:nn {#1} {#2} }
\PrgNewFunction \ClistVarMapInline { M n }
{ \clist_map_inline:Nn #1 {#2} }
\PrgNewFunction \ClistMapVariable { m M n }
{ \clist_map_variable:nNn {#1} #2 {#3} }
\PrgNewFunction \ClistVarMapVariable { M M n }
{ \clist_map_variable:NNn #1 #2 {#3} }
\cs_set_eq:NN \ClistMapBreak \clist_map_break:
\PrgNewConditional \ClistIfExist { M }
{ \clist_if_exist:NTF #1 { \Result { \cTrueBool } } { \Result { \cFalseBool } } }
}
\PrgNewConditional \ClistIfEmpty { m }
{ \clist_if_empty:nTF {#1} { \Result { \cTrueBool } } { \Result { \cFalseBool } } }

\PrgNewConditional \ClistVarIfEmpty { m }
{ \clist_if_empty:NTF #1 { \Result { \cTrueBool } } { \Result { \cFalseBool } } }

\PrgNewConditional \ClistIfIn { m m }
{ \clist_if_in:nnTF {#1} {#2} { \Result { \cTrueBool } } { \Result { \cFalseBool } } }

\PrgNewConditional \ClistVarIfIn { M m }
{ \clist_if_in:NnTF #1 {#2} { \Result { \cTrueBool } } { \Result { \cFalseBool } } }