This package provides an intuitive functional programming interface for LaTeX2, which is an alternative choice to expl3 or LuaTeX, if you want to do programming in LaTeX.

Although there are functions in LaTeX3 programming layer (expl3), 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 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  
1.1 Evaluation from Inside to Outside .......................... 6  
1.2 Group Scoping of Functions ................................. 6  
1.3 Tracing Evaluation of Functions ............................ 7  
1.4 Definitions of Functions ................................... 8  
1.5 Variants of Arguments ....................................... 8  

2 Functional Programming (Prg)  
2.1 Defining Functions and Conditionals ......................... 10  
2.2 Returning Values and Printing Tokens ...................... 11  
2.3 Running Code with Anonymous Functions .................... 11  

3 Argument Using (Use)  
3.1 Evaluating Functions ....................................... 12  
3.2 Expanding Tokens ........................................... 12  
3.3 Using Tokens ................................................. 13  

4 Control Structures (Bool)  
4.1 Constant and Scratch Booleans .............................. 14  
4.2 Boolean Expressions ....................................... 14  
4.3 Creating and Setting Booleans .............................. 15  
4.4 Viewing Booleans ........................................... 15  
4.5 Booleans and Conditionals ................................. 16  
4.6 Booleans and Logical Loops ................................. 17  

5 Token Lists (Tl)  
5.1 Constant and Scratch Token Lists ......................... 19  
5.2 Creating and Using Token Lists ............................ 20  
5.3 Viewing Token Lists ....................................... 21  
5.4 Setting Token List Variables .............................. 21  
5.5 Replacing Tokens ........................................... 22  
5.6 Working with the Content of Token Lists .................. 24  
5.7 Mapping over Token Lists ................................... 25  
5.8 Token List Conditionals .................................... 26  
5.9 Token List Case Functions .................................. 29
10.4 Setting Comma Lists ........................................ 65
10.5 Modifying Comma Lists ........................................ 66
10.6 Working with the Contents of Comma Lists .............. 67
10.7 Comma Lists as Stacks ......................................... 68
10.8 Mapping over Comma Lists .................................... 69
10.9 Comma List Conditionals ...................................... 70

11 Sequences and Stacks (Seq) .................................. 72
11.1 Constant and Scratch Sequences .............................. 72
11.2 Creating and Using Sequences ................................. 72
11.3 Viewing Sequences ............................................ 73
11.4 Setting Sequences ............................................. 73
11.5 Modifying Sequences .......................................... 75
11.6 Working with the Contents of Sequences .................. 75
11.7 Sequences as Stacks ........................................... 76
11.8 Recovering Items from Sequences ............................ 77
11.9 Mapping over Sequences ...................................... 79
11.10 Sequence Conditionals ..................................... 79

12 Property Lists (Prop) ........................................ 81
12.1 Constant and Scratch Sequences .............................. 81
12.2 Creating and Using Property Lists .......................... 81
12.3 Viewing Property Lists ...................................... 82
12.4 Setting Property Lists ....................................... 82
12.5 Recovering Values from Property Lists .................... 84
12.6 Mapping over property lists .................................. 85
12.7 Property List Conditionals ................................... 86

13 Regular Expressions (Regex) .................................. 87
13.1 Regular Expression Variables ................................ 87
13.2 Regular Expression Matching ................................ 88
13.3 Regular Expression Submatch Extraction .................. 90
13.4 Regular Expression Replacement ............................ 92
13.5 Syntax of Regular Expressions ............................... 95
  13.5.1 Regular Expression Examples ............................ 95
  13.5.2 Characters in Regular Expressions ..................... 96
  13.5.3 Characters Classes ...................................... 96
  13.5.4 Structure: Alternatives, Groups, Repetitions ........ 97
  13.5.5 Matching Exact Tokens ................................ 98
  13.5.6 Miscellaneous .......................................... 99
13.6 Syntax of the Replacement Text ............................. 100

14 Token Manipulation (Token) .................................. 102

15 Text Processing (Text) ....................................... 104
15.1 Case Changing .................................................. 104
16 Files (File) 106
   16.1 File Operation Functions .................................... 106

17 Quarks (Quark) 108
   17.1 Constant Quarks .................................................. 108
   17.2 Quark Conditionals ................................................. 108

18 Legacy Concepts (Legacy) 109

19 The Source Code 110
   19.1 Interfaces for Functional Programming (Prg) ..................... 110
      19.1.1 Setting Functional Package ................................ 110
      19.1.2 Creating New Functions and Conditionals .................... 112
      19.1.3 Creating Some Useful Functions .............................. 120
      19.1.4 Return Values and Return Processors ........................ 120
      19.1.5 Evaluating Functions inside Arguments, I ................... 121
      19.1.6 Evaluating Functions inside Arguments, II .................. 122
      19.1.7 Printing Contents to the Input Stream ...................... 123
      19.1.8 Filling Arguments into Inline Commands ..................... 124
      19.1.9 Checking for Local or Global Variables .................... 124
   19.2 Interfaces for Argument Using (Use) ............................ 125
   19.3 Interfaces for Control Structures (Bool) ........................ 126
   19.4 Interfaces for Token Lists (Tl) ................................. 128
   19.5 Interfaces for Strings (Str) ..................................... 132
   19.6 Interfaces for Integers (Int) ................................... 135
   19.7 Interfaces for Floating Point Numbers (Fp) ..................... 138
   19.8 Interfaces for Dimensions (Dim) .................................. 140
   19.9 Interfaces for Sorting Functions (Sort) ........................ 142
   19.10 Interfaces for Comma Separated Lists (Clist) .................. 142
   19.11 Interfaces for Sequences and Stacks (Seq) ........................ 146
   19.12 Interfaces for Property Lists (Prop) .......................... 151
   19.13 Interfaces for Regular Expressions (Regex) .................... 154
   19.14 Interfaces for Token Manipulation (Token) ..................... 158
   19.15 Interfaces for Text Processing (Text) .......................... 159
   19.16 Interfaces for Files (File) ...................................... 160
   19.17 Interfaces for Quarks (Quark) ................................... 161
   19.18 Interfaces to Legacy Concepts (Legacy) ........................ 161
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}}
  \prgReturn {\expValue \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 `\prgReturn` 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 \mathFact { m } {
  \intCompareTF {#1} = {0} {
    \prgReturn {1}
  } {
    \prgReturn {\intEval{#1*\mathFact{\intEval{#1-1}}}}
  }
}
\mathFact{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

1.3 Tracing Evaluation of Functions

Since every function in 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 \Functional{tracing=true}. For example, the tracing log of the first example in this chapter will be the following:

Same as exp13, the names of local variables must start with l, while names of global variables must start with g. The difference is that 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 \intSet{lTmpaInt}{1} and \intSet{gTmpbInt}{2}.

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

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

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

Since 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 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 functional package, there are only three variants (c, e, V) are provided, and these variants are defined as functions (\texttt{\expName}, \texttt{\expWhole}, \texttt{\expValue}, respectively), which are easier to use for regular users.

\begin{verbatim}
\newcommand\test{uvw}
\expName{test} \verb|uvw|

\newcommand\test{uvw}
\expWhole{111\test222} \verb|111uvw222|

\intSet\lTmpaInt{123}
\expValue\lTmpaInt \verb|123|
\end{verbatim}

The most interesting feature is that you can compose these functions. For example, you can easily get the v variant of \texttt{expl3} by simply composing \texttt{\expName} and \texttt{\expValue} functions:

\begin{verbatim}
\intSet\lTmpaInt{123}
\expValue{\expName{\lTmpaInt}} \verb|123|
\end{verbatim}
Chapter 2

Functional Programming (Prg)

2.1 Defining Functions and Conditionals

\prgNewFunction \langle function \rangle \{ \langle argument specification \rangle \} \{ \langle code \rangle \}

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

The \{ \langle argument specification \rangle \} 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 \prgNewFunction, the argument will be evaluated and replaced with the returned value.

\prgSetEqFunction \langle function_1 \rangle \langle function_2 \rangle

Sets \langle function_1 \rangle as an alias of \langle function_2 \rangle.

\prgNewConditional \langle function \rangle \{ \langle argument specification \rangle \} \{ \langle code \rangle \}

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

Assume the \langle function \rangle is \texttt{fooIfBar}, then another three functions are also created at the same time: \texttt{fooIfBarT}, \texttt{fooIfBarF}, and \texttt{fooIfBarTF}. They have extra arguments which are \{ \langle true code \rangle \} or/and \{ \langle false code \rangle \}. For example, if you write

\prgNewConditional \texttt{fooIfBar} \{Mm\} \{ code with return value \texttt{cTrueBool} or \texttt{cFalseBool}\}

Then the following four functions are created:

- \texttt{fooIfBar} \langle arg_1 \rangle \{ \langle arg_2 \rangle \}
- \texttt{fooIfBarT} \langle arg_1 \rangle \{ \langle arg_2 \rangle \} \{ \langle true code \rangle \}
- \texttt{fooIfBarF} \langle arg_1 \rangle \{ \langle arg_2 \rangle \} \{ \langle false code \rangle \}
- \texttt{fooIfBarTF} \langle arg_1 \rangle \{ \langle arg_2 \rangle \} \{ \langle true code \rangle \} \{ \langle false code \rangle \}
2.2 Returning Values and Printing Tokens

Just like LuaTeX, functional package also provides \texttt{\prgReturn} and \texttt{\prgPrint} functions.

\begin{verbatim}
\prgReturn \{\langle tokens\rangle\}
\end{verbatim}

Returns \texttt{\langle tokens\rangle} as result of current function or conditional. This function is normally used in the \texttt{\langle code\rangle} of \texttt{\prgNewFunction} or \texttt{\prgNewConditional}, and it \textbf{must} be the last function evaluated in the \texttt{\langle code\rangle}. If it is missing, the return value of the last function evaluated in the \texttt{\langle code\rangle} is returned. Therefore, the following two examples produce the same output:

\begin{verbatim}
\IgnoreSpacesOn
\prgNewFunction \mathSquare { m } { 
  \intSet \lTmpaInt \{\intEval \{#1 * #1\}\} 
  \prgReturn \{\expValue \lTmpaInt\} 
}\IgnoreSpacesOff
\mathSquare{5}
\end{verbatim}

\begin{verbatim}
\IgnoreSpacesOn
\prgNewFunction \mathSquare { m } { 
  \intSet \lTmpaInt \{\intEval \{#1 * #1\}\} 
  \expValue \lTmpaInt 
}\IgnoreSpacesOff
\mathSquare{5}
\end{verbatim}

Functional package takes care of return values, and only print them to the input stream if the outer most functions are evaluated.

\begin{verbatim}
\prgPrint \{\langle tokens\rangle\}
\end{verbatim}

Prints \texttt{\langle tokens\rangle} directly to the input stream. If there is no function defined with \texttt{\prgNewFunction} in \texttt{\langle tokens\rangle}, you can omit \texttt{\prgPrint} and write only \texttt{\langle tokens\rangle}. But if there is any function defined with \texttt{\prgNewFunction} in \texttt{\langle tokens\rangle}, you \textbf{have to} use \texttt{\prgPrint} function.

2.3 Running Code with Anonymous Functions

\begin{verbatim}
\prgDo \{\langle code\rangle\}
\end{verbatim}

Treats \texttt{\langle code\rangle} as an anonymous function without arguments and evaluates it.

\begin{verbatim}
\prgRunOneArgCode \{\langle arg1\rangle\} \{\langle code\rangle\}
\prgRunTwoArgCode \{\langle arg1\rangle\} \{\langle arg2\rangle\} \{\langle code\rangle\}
\prgRunThreeArgCode \{\langle arg1\rangle\} \{\langle arg2\rangle\} \{\langle arg3\rangle\} \{\langle code\rangle\}
\prgRunFourArgCode \{\langle arg1\rangle\} \{\langle arg2\rangle\} \{\langle arg3\rangle\} \{\langle arg4\rangle\} \{\langle code\rangle\}
\end{verbatim}

Treats \texttt{\langle code\rangle} as an anonymous function with one to four arguments respectively, and evaluates it. In evaluating the \texttt{\langle code\rangle}, functional package first evaluates \texttt{\langle arg1\rangle} to \texttt{\langle arg4\rangle}, then replaces \texttt{\#1} to \texttt{\#4} in \texttt{\langle code\rangle} with the return values respectively.
Chapter 3

Argument Using (Use)

3.1 Evaluating Functions

\evalWhole \{\langle tokens\rangle\}

Evaluates all functions (defined with \prgNewFunction) in \langle tokens\rangle and replaces them with their return values, then returns the resulting tokens.

\tlSet \ltmpaTl \{a\intEval{2*3}b\}
\tlSet \ltmpbTl \{\evalWhole \{a\intEval{2*3}b\}\}

In the above example, \ltmpaTl contains a\intEval{2*3}b, while \ltmpbTl contains a6b.

\evalNone \{\langle tokens\rangle\}

Prevents the evaluation of its argument, returning \langle tokens\rangle without touching them.

\tlSet \ltmpaTl \{\intEval{2*3}\}
\tlSet \ltmpbTl \{\evalNone \{\intEval{2*3}\}\}

In the above example, \ltmpaTl contains 6, while \ltmpbTl contains \intEval{2*3}.

3.2 Expanding Tokens

\expName \{\langle control sequence name\rangle\}

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

\expValue \langle variable\rangle

Recovers the content of a \langle variable\rangle 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 \langle tl var\rangle, or \intUse for \langle int var\rangle.

\expWhole \{\langle tokens\rangle\}

Expands the \langle tokens\rangle exhaustively and returns the result.
\unExpand \{\langle tokens\rangle\}
Prevents expansion of the \langle tokens\rangle inside the argument of \expWhole function. The argument of \unExpand must be surrounded by braces.

\onlyName \{\langle tokens\rangle\}
Expands the \langle tokens\rangle until only characters remain, and then converts this into a control sequence. Further expansion of this control sequence is then inhibited inside the argument of \expWhole function.

\onlyValue \{\langle variable\rangle\}
Recovers the content of the \langle variable\rangle, then prevents expansion of this material inside the argument of \expWhole function.

3.3 Using Tokens

\useOne \{\langle argument\rangle\}
\gobbleOne \{\langle argument\rangle\}
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

\useGobble \{\langle arg1\rangle\} \{\langle arg2\rangle\}
\gobbleUse \{\langle arg1\rangle\} \{\langle arg2\rangle\}
These functions absorb two arguments. The function \useGobble discards the second argument, and returns the content of the first argument. \gobbleUse discards the first argument, and returns the content of the second argument. For example

\useGobble{abc}\{uvw\}\gobbleUse{abc}\{uvw\}
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 \{\prgReturn{True!}\} \{\prgReturn{False!}\}
\boolVarIfTF \cFalseBool \{\prgReturn{True!}\} \{\prgReturn{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 Boolean Expressions

As we have a boolean datatype and predicate functions returning boolean ⟨true⟩ or ⟨false⟩ values, it seems only fitting that we also provide a parser for ⟨boolean expressions⟩.

A boolean expression is an expression which given input in the form of predicate functions and boolean variables, return boolean ⟨true⟩ or ⟨false⟩. It supports the logical operations And, Or and Not as the well-known infix operators && and || and prefix ! with their usual precedences (namely, && binds more tightly than ||). In addition to this, parentheses can be used to isolate sub-expressions. For example,

\intCompare \{1\} = \{1\} &&
(\intCompare \{2\} = \{3\} ||
\intCompare \{4\} < \{4\} ||
\strIfEq \{abc\} \{def\}\)
&&
! \intCompare \{2\} = \{4\}

is a valid boolean expression.
Contrarily to some other programming languages, the operators \&\& and || evaluate both operands in all cases, even when the first operand is enough to determine the result.

### 4.3 Creating and Setting Booleans

\texttt{\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.

\texttt{\boolConst \langle boolean \rangle \lbrace \langle boolexpr \rangle \rbrace}

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

\begin{verbatim}
\boolConst \cFooSomeBool \{\intCompare{3}>{2}\}
\boolVarLog \cFooSomeBool
\end{verbatim}

\texttt{\boolSet \langle boolean \rangle \lbrace \langle boolexpr \rangle \rbrace}

Evaluates the \langle boolean expression \rangle and sets the \langle boolean \rangle variable to the logical truth of this evaluation. For example

\begin{verbatim}
\boolSet \lTmpaBool \{\intCompare{3}<{4}\}
\boolVarLog \lTmpaBool
\end{verbatim}

\begin{verbatim}
\boolSet \lTmpaBool \{\intCompare{3}<{4}\} \&\& \strIfEq{abc}{uvw}
\boolVarLog \lTmpaBool
\end{verbatim}

\texttt{\boolSetTrue \langle boolean \rangle}

Sets \langle boolean \rangle logically true.

\texttt{\boolSetFalse \langle boolean \rangle}

Sets \langle boolean \rangle logically false.

\texttt{\boolSetEq \langle boolean1 \rangle \langle boolean2 \rangle}

Sets \langle boolean1 \rangle to the current value of \langle boolean2 \rangle. For example

\begin{verbatim}
\boolSetTrue \lTmpaBool
\boolSetEq \lTmpbBool \lTmpaBool
\boolVarLog \lTmpbBool
\end{verbatim}

### 4.4 Viewing Booleans

\texttt{\boolLog \lbrace \langle boolean expression \rangle \rbrace}

Writes the logical truth of the \langle boolean expression \rangle in the log file.
\boolVarLog \langle \text{boolean} \rangle

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

\boolShow \{(\text{boolean expression})\}

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

\boolVarShow \langle \text{boolean} \rangle

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

4.5 Booleans and Conditionals

\boolIfExist \langle \text{boolean} \rangle
\boolIfExistT \langle \text{boolean} \rangle \{\langle \text{true code} \rangle\}
\boolIfExistF \langle \text{boolean} \rangle \{\langle \text{false code} \rangle\}
\boolIfExistTF \langle \text{boolean} \rangle \{\langle \text{true code} \rangle\} \{\langle \text{false code} \rangle\}

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

\boolIfExistTF \lTmpaBool \{\prgReturn{Yes}\} \{\prgReturn{No}\} \hspace{1cm} \text{Yes No}
\boolIfExistTF \lFooUndefinedBool \{\prgReturn{Yes}\} \{\prgReturn{No}\}

\boolVarIf \langle \text{boolean} \rangle
\boolVarIfT \langle \text{boolean} \rangle \{\langle \text{true code} \rangle\}
\boolVarIfF \langle \text{boolean} \rangle \{\langle \text{false code} \rangle\}
\boolVarIfTF \langle \text{boolean} \rangle \{\langle \text{true code} \rangle\} \{\langle \text{false code} \rangle\}

Tests the current truth of \langle \text{boolean} \rangle, and continues evaluation based on this result. For example

\boolVarIfTF \lTmpaBool \{\prgReturn{True!}\} \{\prgReturn{False!}\} \hspace{1cm} \text{True! False!}
\boolVarIfTF \lTmpaBool \{\prgReturn{True!}\} \{\prgReturn{False!}\}
\boolSetFalse \lTmpaBool
\boolVarIfTF \lTmpaBool \{\prgReturn{True!}\} \{\prgReturn{False!}\}

\boolVarNot \langle \text{boolean} \rangle
\boolVarNotT \langle \text{boolean} \rangle \{\langle \text{true code} \rangle\}
\boolVarNotF \langle \text{boolean} \rangle \{\langle \text{false code} \rangle\}
\boolVarNotTF \langle \text{boolean} \rangle \{\langle \text{true code} \rangle\} \{\langle \text{false code} \rangle\}

Evaluates \langle \text{true code} \rangle if \langle \text{boolean} \rangle is \text{false}, and \langle \text{false code} \rangle if \langle \text{boolean} \rangle is \text{true}. For example

\boolVarNotTF \{\intCompare{3}>{2}\} \{\prgReturn{Yes}\} \{\prgReturn{No}\} \hspace{1cm} \text{No}

\boolVarAnd \langle \text{boolean}_1 \rangle \langle \text{boolean}_2 \rangle
\boolVarAndT \langle \text{boolean}_1 \rangle \langle \text{boolean}_2 \rangle \{\langle \text{true code} \rangle\}
\boolVarAndF \langle \text{boolean}_1 \rangle \langle \text{boolean}_2 \rangle \{\langle \text{false code} \rangle\}
\boolVarAndTF \langle \text{boolean}_1 \rangle \langle \text{boolean}_2 \rangle \{\langle \text{true code} \rangle\} \{\langle \text{false code} \rangle\}

Implements the “And” operation between two booleans, hence is \text{true} if both are \text{true}. For example
\boolVarAndTF {\intCompare{3}>{2}} {\intIfOdd{6}} {\prgReturn{Yes}} {\prgReturn{No}}

\boolVarOr {boolean_1} {boolean_2}
\boolVarOrT {boolean_1} {boolean_2} {true code}
\boolVarOrF {boolean_1} {boolean_2} {false code}
\boolVarOrTF {boolean_1} {boolean_2} {true code} {false code}

Implements the “Or” operation between two booleans, hence is true if either one is true. For example

\boolVarOrTF {\intCompare{3}>{2}} {\intIfOdd{6}} {\prgReturn{Yes}} {\prgReturn{No}}

\boolVarXor {boolean_1} {boolean_2}
\boolVarXorT {boolean_1} {boolean_2} {true code}
\boolVarXorF {boolean_1} {boolean_2} {false code}
\boolVarXorTF {boolean_1} {boolean_2} {true code} {false code}

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

\boolVarXorTF {\intCompare{3}>{2}} {\intIfOdd{6}} {\prgReturn{Yes}} {\prgReturn{No}}

4.6 Booleans and Logical Loops

Loops using either boolean expressions or stored boolean values.

\boolVarDoUntil {boolean} {code}

Places the \textit{(code)} in the input stream for \LaTeX{} to process, and then checks the logical value of the \textit{(boolean)}. If it is false then the \textit{(code)} is inserted into the input stream again and the process loops until the \textit{(boolean)} is true.

\IgnoreSpacesOn
\boolSetFalse \lTmpaBool
\intZero \lTmpaInt
\clistClear \lTmpaClist
\boolVarDoUntil \lTmpaBool {
  \intIncr \lTmpaInt
  \clistPutRight \lTmpaClist {\expValue\lTmpaInt}
  \intCompare \lTmpaInt = {10} {\boolSetTrue \lTmpaBool}
} \clistVarJoin \lTmpaClist {;}
\IgnoreSpacesOff

\boolVarDoWhile {boolean} {code}

Places the \textit{(code)} in the input stream for \LaTeX{} to process, and then checks the logical value of the \textit{(boolean)}. If it is true then the \textit{(code)} is inserted into the input stream again and the process loops until the \textit{(boolean)} is false.
This function first checks the logical value of the \( boolean \). If it is \texttt{false} the \( code \) is placed in the input stream and expanded. After the completion of the \( code \) the truth of the \( boolean \) is re-evaluated. The process then loops until the \( boolean \) is \texttt{true}.

This function first checks the logical value of the \( boolean \). If it is \texttt{true} the \( code \) is placed in the input stream and expanded. After the completion of the \( code \) the truth of the \( boolean \) is re-evaluated. The process then loops until the \( boolean \) is \texttt{false}.
Chapter 5

Token Lists (Tl)

TEX works with tokens, and \LaTeX{} therefore provides a number of functions to deal with lists of tokens. Token lists may be present directly in the argument to a function:

\tlFoo {a collection of \tokens}

or may be stored in a so-called “token list variable”, which have the suffix Tl: a token list variable can also be used as the argument to a function, for example

\tlVarFoo \lSomeTl

In both cases, functions are available to test and manipulate the lists of tokens, and these have the module prefix Tl. In many cases, functions which can be applied to token list variables are paired with similar functions for application to explicit lists of tokens: the two “views” of a token list are therefore collected together here.

A token list (explicit, or stored in a variable) can be seen either as a list of “items”, or a list of “tokens”. An item is whatever \useOne{} would grab as its argument: a single non-space token or a brace group, with optional leading explicit space characters (each item is thus itself a token list). A token is either a normal N argument, or \␣, {, or } (assuming normal TEX category codes). Thus for example

{Hello} world

contains 6 items (Hello, w, o, r, l and d), but 13 tokens ({, H, e, l, 1, o, }, , v, o, r, l and d). Functions which act on items are often faster than their analogue acting directly on tokens.

5.1 Constant and Scratch Token Lists

\cSpaceTl

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

\cEmptyTl

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.
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.

\tlNew \lFooSomeTl

\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.

\tlConst \cFooSomeTl \{abc\}

\tlUse \langle tl var \rangle

Recovers 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.

\tlUse \lTmpbTl

\tlToStr \{ \langle token list \rangle \}

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

\tlToStr \{12\abc34\}

\tlVarToStr \langle tl var \rangle

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

\tlSet \lTmpaTl \{12\abc34\}
\tlVarToStr \lTmpaTl
5.3 Viewing Token Lists

\tlLog \{token list\}

Writes the \textit{(token list)} in the log file. See also \tlShow which displays the result in the terminal.

\tlLog \{123\abc456\}

\tlVarLog \{tl var\}

Writes the content of the \textit{(tl var)} in the log file. See also \tlVarShow which displays the result in the terminal.

\tlSet \lTmpaTl \{123\abc456\}
\tlVarLog \lTmpaTl

\tlShow \{token list\}

Displays the \textit{(token list)} on the terminal.

\tlShow \{123\abc456\}

\tlVarShow \{tl var\}

Displays the content of the \textit{(tl var)} on the terminal.

\tlSet \lTmpaTl \{123\abc456\}
\tlVarShow \lTmpaTl

5.4 Setting Token List Variables

\tlSet \{tl var\} \{tokens\}

Sets \textit{(tl var)} to contain \textit{(tokens)}, removing any previous content from the variable.

\tlSet \lTmpbTl \{\intMathMult{4}{5}\}
\tlUse \lTmpbTl

\tlSetEq \{tl var\}_1 \{tl var\}_2

Sets the content of \textit{(tl var\textsubscript{1})} equal to that of \textit{(tl var\textsubscript{2})}.

\tlSet \lTmpaTl \{abc\}
\tlSetEq \lTmpbTl \lTmpaTl
\tlUse \lTmpbTl

\tlClear \{tl var\}

Clears all entries from the \textit{(tl var)}. 
\tlSet \ltmpjTl \{One\}
\tlClear \ltmpjTl
\tlSet \ltmpjTl \{Two\}
\tlUse \ltmpjTl

\tlClearNew \tlvar

Ensures that the \(\tlvar\) exists globally by applying \tlNew if necessary, then applies \tlClear to leave the \(\tlvar\) empty.

\tlClearNew \lfooSomeTl

\tlConcat \tlvar\tlvar\tlvar

Concatenates the content of \(\tlvar\tlvar\) together and saves the result in \(\tlvar\). The \(\tlvar\tlvar\) is placed at the left side of the new token list.

\tlSet \ltmpbTl \{con\}
\tlSet \ltmpcTl \{cat\}
\tlConcat \ltmpaTl \ltmpbTl \ltmpcTl
\tlUse \ltmpaTl

\tlPutLeft \tlvar \{\tokens\}

Appends \(\tokens\) to the left side of the current content of \(\tlvar\).

\tlSet \ltmpkTl \{Functional\}
\tlPutLeft \ltmpkTl \{Hello\}
\tlUse \ltmpkTl

\tlPutRight \tlvar \{\tokens\}

Appends \(\tokens\) to the right side of the current content of \(\tlvar\).

\tlSet \ltmpkTl \{Functional\}
\tlPutRight \ltmpkTl \{World\}
\tlUse \ltmpkTl

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).

\tlVarReplaceOnce \tlvar \{old tokens\} \{new tokens\}

Replaces the first (leftmost) occurrence of \(old tokens\) in the \(\tlvar\) 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).
\tlSet \lTmpaTl {1(bc)2bc3} \tlVarReplaceOnce \lTmpaTl {bc} {xx} \tlUse \lTmpaTl

Replaces all occurrences of \(\text{old tokens}\) in the \(\tl var\) with \(\text{new tokens}\). \(\text{Old tokens}\) 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 \(\text{old tokens}\) may remain after the replacement (see \tlVarRemoveAll for an example).

\tlSet \lTmpaTl {1(bc)2bc3} \tlVarRemoveAll \lTmpaTl {bc} \tlUse \lTmpaTl

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

\tlSet \lTmpaTl {abbccd} \tlVarRemoveAll \lTmpaTl {bc} \tlUse \lTmpaTl

Removes all occurrences of \(\text{tokens}\) from the \(\tl var\). \(\text{Tokens}\) 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 \(\text{tokens}\) may remain after the removal, for instance,

\tlTrimSpaces \{\text{token list}\}

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

\tlVarTrimSpaces \tlUse \lTmpaTl

Sets the \(\tl var\) 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

\texttt{\textbackslash tlCount \{tokens\}}

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

\begin{verbatim}
\tlCount \{12\abc34\}
\end{verbatim}

\texttt{\textbackslash tlVarCount \langle tl var \rangle}

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

\begin{verbatim}
\tlSet \lTmpaTl \{12\abc34\}
\tlVarCount \lTmpaTl
\end{verbatim}

\texttt{\textbackslash tlHead \{\textbackslash token list\}}

Returns the first \texttt{\langle item \rangle} in the \texttt{\langle token list \rangle}, discarding the rest of the \texttt{\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{  abc }2}
\end{verbatim}

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

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

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

\texttt{\textbackslash tlVarHead \langle tl var \rangle}

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

\begin{verbatim}
\tlSet \lTmpaTl \{HELLO\}
\tlVarHead \lTmpaTl
\end{verbatim}

\texttt{\textbackslash tlTail \{\textbackslash token list\}}

Discards all leading explicit space characters (explicit tokens with character code 32 and category code 10) and the first \texttt{\langle item \rangle} in the \texttt{\langle 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 \texttt{\\{bc\}\_d}. A blank \texttt{\langle token list \rangle} (see \texttt{\textbackslash tlIfBlank}) results in \texttt{\tlTail} returning nothing.
\tlVarTail \langle tl var \rangle

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

\tlSet \lTmpaTl \{HELLO\}  \tlVarTail \lTmpaTl  \begin{array}{l}  \text{ELLO}  \end{array}

\tlItem \{\langle token list \rangle\} \{ \langle integer expression \rangle \}  
\tlVarItem \langle tl var \rangle \{ \langle integer expression \rangle \}

Indexing items in the \langle token list \rangle from 1 on the left, this function evaluates the \langle integer expression \rangle and returns the appropriate item from the \langle token list \rangle. If the \langle 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.

\tlItem \{abcd\} \{3\}  \begin{array}{l}  \text{c}  \end{array}

\tlRandItem \{\langle token list \rangle\}  
\tlVarRandItem \langle tl var \rangle \{\langle token list \rangle\}

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

\tlRandItem \{abcdef\}  \tlRandItem \{abcdef\}  \begin{array}{l}  \text{c b}  \end{array}

### 5.7 Mapping over Token Lists

\tlMapInline \{\langle token list \rangle\} \{ \langle inline function \rangle \}

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

\IgnoreSpacesOn  \tlClear \lTmpaTl  \tlMapInline \{\langle token list \rangle\} \{ \langle inline function \rangle \}  \begin{array}{l}  \text{[o][n][c]}  \end{array}

\tlMapInline \langle tl var \rangle \{ \langle inline function \rangle \}

Applies the \langle inline function \rangle to every \langle item \rangle stored within the \langle tl var \rangle. The \langle inline function \rangle should consist of code which receives the \langle item \rangle as \#1.
5.8 Token List Conditionals

\tlIfExist \langle tl \ var \rangle \ {\langle (true \ code) \rangle \ {\langle (false \ code) \rangle \ )}

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

\tlIfExistTF \langle tl \ var \rangle \ {\langle prgReturn{Yes} \rangle \ {\langle prgReturn{No} \rangle \ )}
\tlIfExistTF \langle tl \ FooUndefined \rangle \ {\langle prgReturn{Yes} \rangle \ {\langle prgReturn{No} \rangle \ )

Yes No
\texttt{\texttt{\tlIfEmpty} \langle \text{token list} \rangle} \\texttt{\{\texttt{true code}\}} \\texttt{\{\texttt{false code}\}}

Tests if the (\textit{token list}) is entirely empty (\textit{i.e.} contains no tokens at all). For example

\begin{verbatim}
\tlIfEmptyTF {abc} \{\prgReturn{Empty}} \{\prgReturn{NonEmpty}}
\tlIfEmptyTF {} \{\prgReturn{Empty}} \{\prgReturn{NonEmpty}}
\end{verbatim}

\texttt{\texttt{\tlVarIfEmpty} \langle \text{tl var} \rangle} \\texttt{\{\texttt{true code}\}} \\texttt{\{\texttt{false code}\}}

Tests if the (\textit{token list variable}) is entirely empty (\textit{i.e.} contains no tokens at all). For example

\begin{verbatim}
\tlSet \lTmpaTl {abc}
\tlVarIfEmptyTF \lTmpaTl \{\prgReturn{Empty}} \{\prgReturn{NonEmpty}}
\tlClear \lTmpaTl
\tlVarIfEmptyTF \lTmpaTl \{\prgReturn{Empty}} \{\prgReturn{NonEmpty}}
\end{verbatim}

\texttt{\texttt{\tlIfBlank} \langle \text{token list} \rangle} \\texttt{\{\texttt{true code}\}} \\texttt{\{\texttt{false code}\}}

Tests if the (\textit{token list}) consists only of blank spaces (\textit{i.e.} contains no item). The test is \texttt{true} if \langle \text{token list} \rangle is zero or more explicit space characters (explicit tokens with character code 32 and category code 10), and is \texttt{false} otherwise.

\begin{verbatim}
\tlIfBlankTF { } \{\prgReturn{Yes}} \{\prgReturn{No}}
\end{verbatim}

\texttt{\texttt{\tlIfEq} \langle \text{token list 1} \rangle \{\texttt{true code}\}} \\texttt{\{\texttt{false code}\}}

Tests if (\textit{token list 1}) and (\textit{token list 2}) contain the same list of tokens, both in respect of character codes and category codes. See \texttt{\strIfEq} if category codes are not important. For example

\begin{verbatim}
\tlIfEqTF {abc} {abc} \{\prgReturn{Yes}} \{\prgReturn{No}}
\tlIfEqTF {abc} {xyz} \{\prgReturn{Yes}} \{\prgReturn{No}}
\end{verbatim}

\texttt{\texttt{\tlVarIfEq} \langle \text{tl var 1} \rangle \langle \text{tl var 2} \rangle} \\texttt{\{\texttt{true code}\}} \\texttt{\{\texttt{false code}\}}

Compares the content of two (\textit{token list variables}) and is logically \texttt{true} if the two contain the same list of tokens (\textit{i.e.} identical in both the list of characters they contain and the category codes of those characters). For example
CHAPTER 5. TOKEN LISTS (TL) 28

\tlSet \lTmpaTl {abc}
\tlSet \lTmpbTl {abc}
\tlSet \lTmpcTl {xyz}
\tlVarIfEqTF \lTmpaTl \lTmpbTl {\prgReturn{Yes}} {\prgReturn{No}}
\tlVarIfEqTF \lTmpaTl \lTmpcTl {\prgReturn{Yes}} {\prgReturn{No}}

Yes No

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

\tlIfIn {⟨token list1⟩} {⟨token list2⟩}
\tlIfInT {⟨token list1⟩} {⟨token list2⟩} {⟨true code⟩}
\tlIfInF {⟨token list1⟩} {⟨token list2⟩} {⟨false code⟩}
\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.

\tlIfInTF {hello world} {o} {\prgReturn{Yes}} {\prgReturn{No}}
\tlIfInTF {hello world} {a} {\prgReturn{Yes}} {\prgReturn{No}}

Yes No

\tlVarIfIn {⟨tl var⟩} {⟨token list⟩}
\tlVarIfInT {⟨tl var⟩} {⟨token list⟩} {⟨true code⟩}
\tlVarIfInF {⟨tl var⟩} {⟨token list⟩} {⟨false code⟩}
\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).

\tlSet \lTmpaTl {hello world}
\tlVarIfInTF \lTmpaTl {o} {\prgReturn{Yes}} {\prgReturn{No}}
\tlVarIfInTF \lTmpaTl {a} {\prgReturn{Yes}} {\prgReturn{No}}

Yes No

\tlIfSingle {⟨token list⟩}
\tlIfSingleT {⟨token list⟩} {⟨true code⟩}
\tlIfSingleF {⟨token list⟩} {⟨false code⟩}
\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.

\tlIfSingleTF {a} {\prgReturn{Yes}} {\prgReturn{No}}
\tlIfSingleTF {abc} {\prgReturn{Yes}} {\prgReturn{No}}

Yes No

\tlVarIfSingle {⟨tl var⟩}
\tlVarIfSingleT {⟨tl var⟩} {⟨true code⟩}
\tlVarIfSingleF {⟨tl var⟩} {⟨false code⟩}
\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 \langle test token list variable \rangle \\
{ \\
  \langle token list variable case \rangle \{ \langle code case \rangle \} \\
  \langle token list variable case \rangle \{ \langle code case \rangle \} \\
  \ldots \\
  \langle token list variable case \rangle \{ \langle code case \rangle \} 
}

This function compares the \langle test token list variable \rangle in turn with each of the \langle token list variable cases \rangle. If the two are equal (as described for \tlVarIfEq) then the associated \langle code \rangle is left in the input stream and other cases are discarded. The function does nothing if there is no match.

\tlVarCaseT \langle test token list variable \rangle \\
{ \\
  \langle token list variable case \rangle \{ \langle code case \rangle \} \\
  \langle token list variable case \rangle \{ \langle code case \rangle \} \\
  \ldots \\
  \langle token list variable case \rangle \{ \langle code case \rangle \} \\
  \{ \langle true code \rangle \} 
}

This function compares the \langle test token list variable \rangle in turn with each of the \langle token list variable cases \rangle. If the two are equal (as described for \tlVarIfEq) then the associated \langle code \rangle is left in the input stream and other cases are discarded. If any of the cases are matched, the \langle true code \rangle is also inserted into the input stream (after the code for the appropriate case).
CHAPTER 5. TOKEN LISTS (TL)

\IgnoreSpacesOn
\tlSet \lTmpaTl {a}
\tlSet \lTmpbTl {b}
\tlSet \lTmpcTl {c}
\tlSet \lTmpkTl {b}
\tlVarCaseT \lTmpkTl {
  \lTmpaTl {\intSet \lTmpkInt {1}}
  \lTmpbTl {\intSet \lTmpkInt {2}}
  \lTmpcTl {\intSet \lTmpkInt {3}}
  }{\prgReturn {\intUse \lTmpkInt}}
\IgnoreSpacesOff

\tlVarCaseF \test token list variable{
  \token list variable case 1 \{\code case 1\}\}
  \token list variable case 2 \{\code case 2\}
  ...
  \token list variable case n \{\code case n\}
  }{\{\false code\}}

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. If none match then the \false code is inserted into the input stream (after the code for the appropriate case).

\IgnoreSpacesOn
\tlSet \lTmpaTl {a}
\tlSet \lTmpbTl {b}
\tlSet \lTmpcTl {c}
\tlSet \lTmpkTl {b}
\tlVarCaseF \lTmpkTl{
  \lTmpaTl {\prgReturn {First}}
  \lTmpbTl {\prgReturn {Second}}
  \lTmpcTl {\prgReturn {Third}}
  }{\prgReturn {No-Match!}}
\IgnoreSpacesOff

\tlVarCaseTF \test token list variable{
  \token list variable case 1 \{\code case 1\}\}
  \token list variable case 2 \{\code case 2\}
  ...
  \token list variable case n \{\code case n\}
  }{\{\true code\}
  {\false code\}}

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. If any of the cases are matched, the \true code is also inserted into the input stream (after the code for the appropriate case), while if none match then the \false code is inserted.
The function \texttt{t1VarCase}, which does nothing if there is no match, is also available.

\begin{verbatim}
\IgnoreSpacesOn
\tlSet \lTmpaTl {a}
\tlSet \lTmpbTl {b}
\tlSet \lTmpcTl {c}
\tlSet \lTmpkTl {b}
\tlVarCaseTF \lTmpkTl {
   \lTmpaTl {\intSet \lTmpkInt {1}}
   \lTmpbTl {\intSet \lTmpkInt {2}}
   \lTmpcTl {\intSet \lTmpkInt {3}}
}
   \prgReturn {\intUse \lTmpkInt}
}{
   \prgReturn {0}
}
\IgnoreSpacesOff
\end{verbatim}
Chapter 6

Strings (Str)

\TeX\ associates each character with a category code: as such, there is no concept of a “string” as commonly understood in many other programming languages. However, there are places where we wish to manipulate token lists while in some sense “ignoring” category codes: this is done by treating token lists as strings in a \TeX\ sense.

A \TeX\ string (and thus an expl3 string) is a series of characters which have category code 12 (“other”) with the exception of space characters which have category code 10 (“space”). Thus at a technical level, a \TeX\ string is a token list with the appropriate category codes. In this documentation, these are simply referred to as strings.

String variables are simply specialised token lists, but by convention should be named with the suffix Str. Such variables should contain characters with category code 12 (other), except spaces, which have category code 10 (blank space). All the functions in this module which accept a token list argument first convert it to a string using \tlToStr\ for internal processing, and do not treat a token list or the corresponding string representation differently.

As a string is a subset of the more general token list, it is sometimes unclear when one should be used over the other. Use a string variable for data that isn’t primarily intended for typesetting and for which a level of protection from unwanted expansion is suitable. This data type simplifies comparison of variables since there are no concerns about expansion of their contents.

6.1 Constant and Scratch Strings

\begin{itemize}
  \item \cAmpersandStr, \cAtsignStr, \cBackslashStr, \cLeftBraceStr, \cRightBraceStr
  \item \cCircumflexStr, \cColonStr, \cDollarStr, \cHashStr, \cPercentStr, \cTildeStr
  \item \cUnderscoreStr, \cZeroStr
\end{itemize}

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

\begin{itemize}
  \item \lTmpaStr, \lTmpbStr, \lTmpcStr, \lTmpiStr, \lTmpjStr, \lTmpkStr
\end{itemize}

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.

\begin{itemize}
  \item \gTmpaStr, \gTmpbStr, \gTmpcStr, \gTmpiStr, \gTmpjStr, \gTmpkStr
\end{itemize}

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

\texttt{\texttt{strNew} \langle str \ var \rangle}

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

\begin{verbatim}
strNew \lFooSomeStr
\end{verbatim}

\texttt{\texttt{strConst} \langle str \ var \rangle \{\langle token \ list \rangle\}}

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

\begin{verbatim}
strConst \cFooSomeStr \{12\abc34\}
\end{verbatim}

\texttt{\texttt{strUse} \langle str \ var \rangle}

Recovers the content of a \( \langle str \ var \rangle \) 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 \( \langle str \rangle \) directly without an accessor function.

\begin{verbatim}
strUse \lTmpaStr
\end{verbatim}

6.3 Viewing Strings

\texttt{\texttt{strLog} \{\langle token \ list \rangle\}}

Writes \( \langle token \ list \rangle \) in the log file.

\begin{verbatim}
strLog \{1234\abcd5678\}
\end{verbatim}

\texttt{\texttt{strVarLog} \langle str \ var \rangle}

 Writes the content of the \( \langle str \ var \rangle \) in the log file.

\begin{verbatim}
strSet \lTmpiStr \{1234\abcd5678\}
\end{verbatim}

\begin{verbatim}
strVarLog \lTmpiStr
\end{verbatim}

\texttt{\texttt{strShow} \{\langle token \ list \rangle\}}

Displays \( \langle token \ list \rangle \) on the terminal.

\begin{verbatim}
strShow \{1234\abcd5678\}
\end{verbatim}

\texttt{\texttt{strVarShow} \langle str \ var \rangle}

Displays the content of the \( \langle str \ var \rangle \) on the terminal.
6.4 Setting String Variables

\texttt{\textbackslash strSet} \langle \texttt{str var} \rangle \{\langle \texttt{token list} \rangle\}

Converts the \langle \texttt{token list} \rangle to a \langle \texttt{string} \rangle, and stores the result in \langle \texttt{str var} \rangle.

\begin{verbatim}
\strSet \lTmpiStr \{
1234\abcd5678
\}
\strShow \lTmpiStr
\end{verbatim}

\texttt{\textbackslash strSetEq} \langle \texttt{str var}_1 \rangle \langle \texttt{str var}_2 \rangle

Sets the content of \langle \texttt{str var}_1 \rangle equal to that of \langle \texttt{str var}_2 \rangle.

\begin{verbatim}
\strSet \lTmpaStr \{abc\}
\strSetEq \lTmpbStr \lTmpaStr
\strUse \lTmpbStr
\end{verbatim}

\texttt{\textbackslash strClear} \langle \texttt{str var} \rangle

Clears the content of the \langle \texttt{str var} \rangle. For example

\begin{verbatim}
\strSet \lTmpjStr \{One\}
\strClear \lTmpjStr
\strSet \lTmpjStr \{Two\}
\strUse \lTmpjStr
\end{verbatim}

\texttt{\textbackslash strClearNew} \langle \texttt{str var} \rangle

Ensures that the \langle \texttt{str var} \rangle exists globally by applying \texttt{\textbackslash strNew} if necessary, then applies \texttt{\textbackslash strClear} to leave the \langle \texttt{str var} \rangle empty.

\begin{verbatim}
\strClearNew \lFooSomeStr
\strUse \lFooSomeStr
\end{verbatim}

\texttt{\textbackslash strConcat} \langle \texttt{str var}_1 \rangle \langle \texttt{str var}_2 \rangle \langle \texttt{str var}_3 \rangle

Concatenates the content of \langle \texttt{str var}_2 \rangle and \langle \texttt{str var}_3 \rangle together and saves the result in \langle \texttt{str var}_1 \rangle. The \langle \texttt{str var}_2 \rangle is placed at the left side of the new string variable. The \langle \texttt{str var}_2 \rangle and \langle \texttt{str var}_3 \rangle must indeed be strings, as this function does not convert their contents to a string.

\begin{verbatim}
\strSet \lTmpbStr \{con\}
\strSet \lTmpcStr \{cat\}
\strConcat \lTmpaStr \lTmpbStr \lTmpcStr
\strUse \lTmpaStr
\end{verbatim}
CHAPTER 6. STRINGS (STR)

\texttt{\textbackslash strPutLeft \{ (\texttt{str} var) \} \{ (\texttt{token list}) \}}

Converts the \( (\texttt{token list}) \) to a \( (\texttt{string}) \), and prepends the result to \( (\texttt{str} \ \texttt{var}) \). The current contents of the \( (\texttt{str} \ \texttt{var}) \) are not automatically converted to a string.

\begin{verbatim}
\strSet \lTmpkStr {Functional}
\strPutLeft \lTmpkStr {Hello}
\strUse \lTmpkStr
\end{verbatim}

HelloFunctional

\texttt{\textbackslash strPutRight \{ (\texttt{str} var) \} \{ (\texttt{token list}) \}}

Converts the \( (\texttt{token list}) \) to a \( (\texttt{string}) \), and appends the result to \( (\texttt{str} \ \texttt{var}) \). The current contents of the \( (\texttt{str} \ \texttt{var}) \) are not automatically converted to a string.

\begin{verbatim}
\strSet \lTmpkStr {Functional}
\strPutRight \lTmpkStr {World}
\strUse \lTmpkStr
\end{verbatim}

FunctionalWorld

\section{6.5 Modifying String Variables}

\texttt{\textbackslash strVarReplaceOnce \{ (\texttt{str} var) \} \{ (\texttt{old}) \} \{ (\texttt{new}) \}}

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

\begin{verbatim}
\strSet \lTmpaStr \{a{bc}bcd\}
\strVarReplaceOnce \lTmpaStr \{bc\} \{xx\}
\strUse \lTmpaStr
\end{verbatim}

a{xx}bcd

\texttt{\textbackslash strVarReplaceAll \{ (\texttt{str} var) \} \{ (\texttt{old}) \} \{ (\texttt{new}) \}}

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

\begin{verbatim}
\strSet \lTmpaStr \{a{bc}bcd\}
\strVarReplaceAll \lTmpaStr \{bc\} \{xx\}
\strUse \lTmpaStr
\end{verbatim}

a{xx}xxd

\texttt{\textbackslash strVarRemoveOnce \{ (\texttt{str} var) \} \{ (\texttt{token list}) \}}

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

\begin{verbatim}
\strSet \lTmpaStr \{a{bc}bcd\}
\strVarRemoveOnce \lTmpaStr \{bc\}
\strUse \lTmpaStr
\end{verbatim}

a{}bcd

\texttt{\textbackslash strVarRemoveAll \{ (\texttt{str} var) \} \{ (\texttt{token list}) \}}

Converts the \( (\texttt{token list}) \) to a \( (\texttt{string}) \) then removes all occurrences of \( (\texttt{token list}) \) from the \( (\texttt{str} \ \texttt{var}) \). As this
function operates from left to right, the pattern \textlangle string\textrangle may remain after the removal, for instance,

\begin{verbatim}
\strSet \lTmpaStr {abccd}
\strVarRemoveAll \lTmpaStr {bc}
\tlUse \lTmpaStr
\end{verbatim}

\texttt{abcd}

\section*{6.6 Working with the Content of Strings}

\textbf{\strCount \{\langle token list\rangle\}}

Returns the number of characters in the string representation of \textlangle token list\textrangle, as an integer denotation. All characters including spaces are counted.

\begin{verbatim}
\strCount {12\abc34}
\end{verbatim}

\texttt{9}

\textbf{\strVarCount \langle tl var\rangle}

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

\begin{verbatim}
\strSet \lTmpaStr {12\abc34}
\strVarCount \lTmpaStr
\end{verbatim}

\texttt{9}

\textbf{\strHead \{\langle token list\rangle\}}

Converts the \textlangle token list\textrangle into a \textlangle string\textrangle. The first character in the \textlangle string\textrangle 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 \textlangle string\textrangle is empty, then nothing is returned.

\begin{verbatim}
\strHead \{HELLO\}
\end{verbatim}

\texttt{H}

\textbf{\strVarHead \langle tl var\rangle}

Converts the \textlangle tl var\textrangle into a \textlangle string\textrangle. The first character in the \textlangle string\textrangle 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 \textlangle string\textrangle is empty, then nothing is returned.

\begin{verbatim}
\strSet \lTmpaStr \{HELLO\}
\strVarHead \lTmpaStr
\end{verbatim}

\texttt{H}

\textbf{\strTail \{\langle token list\rangle\}}

Converts the \textlangle token list\textrangle to a \textlangle string\textrangle, 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 \textlangle token list\textrangle is empty, then nothing is left on the input stream.

\begin{verbatim}
\strTail \{HELLO\}
\end{verbatim}

\texttt{ELLO}
\textbf{\strVarTail\langle tl\ var\rangle}

Converts the \langle tl\ var\rangle\ to a \langle\text{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.

\begin{verbatim}
\strSet \lTmpaStr {HELLO}
\strVarTail \lTmpaStr
\end{verbatim}

\textbf{\strItem\{(token\ list)\}{(integer\ expression)\}}

Converts the \langle token\ list\rangle\ to a \langle\text{string}\rangle, and returns the character in position \langle integer\ expression\rangle\ of the \langle\text{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\text{string}\rangle. Hence, \(-1\) is the right-most character, \textit{etc}.

\begin{verbatim}
\strItem {abcd} {3}
\end{verbatim}

\textbf{\strVarItem\langle tl\ var\rangle\{(integer\ expression)\}}

Converts the \langle tl\ var\rangle\ to a \langle\text{string}\rangle, and returns the character in position \langle integer\ expression\rangle\ of the \langle\text{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\text{string}\rangle. Hence, \(-1\) is the right-most character, \textit{etc}.

\begin{verbatim}
\strSet \lTmpaStr {abcd}
\strVarItem \lTmpaStr {3}
\end{verbatim}

6.7 Mapping over Strings

\textbf{\strMapInline\{(token\ list)\}{(inline\ function)\}}

\textbf{\strVarMapInline\langle\text{str var}\rangle\{(inline\ function)\}}

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

\begin{verbatim}
\IgnoreSpacesOn
\strClear \lTmpaStr
\strMapInline {one} {
   \strPutRight \lTmpaStr {[#1]}
}
\strUse \lTmpaStr
\IgnoreSpacesOff
\end{verbatim}

\textbf{\strMapVariable\{(token\ list)\}{\langle variable\rangle\{(code)\}}

\textbf{\strVarMapVariable\langle\text{str var}\rangle\langle variable\rangle\{(code)\}}

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

\texttt{\textbackslash{}IgnoreSpacesOn}
\texttt{\strClear \lTmaStr}
\texttt{\strMapVariable \{one\} \lTmpiStr { \strPutRight \lTmaStr \{\expWhole { {\lTmpiStr} } \} }}
\texttt{\strUse \lTmaStr}
\texttt{\IgnoreSpacesOff}

Tests whether the \texttt{⟨str var⟩} is currently defined. This does not check that the \texttt{⟨str var⟩} really is a string.

\texttt{\strIfExistT \lTmpaStr \{\prgReturn{Yes} \} \{\prgReturn{No}\}}
\texttt{\strIfExistF \lTmpaStr \{\prgReturn{Yes} \} \{\prgReturn{No}\}}
\texttt{\strIfExistTF \lTmpaStr \{\prgReturn{Yes} \} \{\prgReturn{No}\}}

Tests if the \texttt{⟨string variable⟩} is entirely empty \textit{i.e.} contains no characters at all.

\texttt{\strVarIfEmpty \lTmaStr \{\prgReturn{Empty} \} \{\prgReturn{NonEmpty}\}}
\texttt{\strVarIfEmptyT \lTmaStr \{\prgReturn{Empty} \} \{\prgReturn{NonEmpty}\}}
\texttt{\strVarIfEmptyF \lTmaStr \{\prgReturn{Empty} \} \{\prgReturn{NonEmpty}\}}
\texttt{\strVarIfEmptyTF \lTmaStr \{\prgReturn{Empty} \} \{\prgReturn{NonEmpty}\}}

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

\texttt{\strIfEqTF {abc} {abc} \{\prgReturn{Yes} \} \{\prgReturn{No}\}}
\texttt{\strIfEqTF {abc} {xyz} \{\prgReturn{Yes} \} \{\prgReturn{No}\}}

\texttt{\strVarIfEq \lTmpaStr \lTmpbStr \{\prgReturn{Yes} \} \{\prgReturn{No}\}}
\texttt{\strVarIfEqT \lTmpaStr \lTmpbStr \{\prgReturn{Yes} \} \{\prgReturn{No}\}}
\texttt{\strVarIfEqF \lTmpaStr \lTmpbStr \{\prgReturn{Yes} \} \{\prgReturn{No}\}}
\texttt{\strVarIfEqTF \lTmpaStr \lTmpbStr \{\prgReturn{Yes} \} \{\prgReturn{No}\}}

Compares the content of two \texttt{⟨str variables⟩} and is logically \texttt{true} if the two contain the same characters
in the same order. See `\texttt{\textnormal{t1VarIfEq}}` to compare tokens (including their category codes) rather than characters.

\begin{verbatim}
\strSet {lTmpaStr} {abc}
\strSet {lTmpbStr} {abc}
\strSet {lTmpcStr} {xyz}
\strVarIfEqTF {lTmpaStr} {lTmpbStr} {prgReturn{Yes}} {prgReturn{No}}
\strVarIfEqTF {lTmpaStr} {lTmpcStr} {prgReturn{Yes}} {prgReturn{No}}
\end{verbatim}

\begin{verbatim}
\strIfIn {⟨tl1⟩} {⟨tl2⟩} {prgReturn{true code}} {prgReturn{false code}}
\strIfInT {⟨tl1⟩} {⟨tl2⟩} {prgReturn{true code}} {prgReturn{false code}}
\strIfInF {⟨tl1⟩} {⟨tl2⟩} {prgReturn{true code}} {prgReturn{false code}}
\strIfInTF {⟨tl1⟩} {⟨tl2⟩} {prgReturn{true code}} {prgReturn{false code}}
\end{verbatim}

Converts both (token lists) to (strings) and tests whether (string2) is found inside (string1).

\begin{verbatim}
\strIfInTF {hello world} {o} {prgReturn{Yes}} {prgReturn{No}}
\strIfInTF {hello world} {a} {prgReturn{Yes}} {prgReturn{No}}
\end{verbatim}

\begin{verbatim}
\strVarIfIn {str var} {⟨token list⟩} {prgReturn{true code}} {prgReturn{false code}}
\strVarIfInT {str var} {⟨token list⟩} {prgReturn{true code}} {prgReturn{false code}}
\strVarIfInF {str var} {⟨token list⟩} {prgReturn{true code}} {prgReturn{false code}}
\strVarIfInTF {str var} {⟨token list⟩} {prgReturn{true code}} {prgReturn{false code}}
\end{verbatim}

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

\begin{verbatim}
\strVarIfInTF {lTmpaStr} {o} {prgReturn{Yes}} {prgReturn{No}}
\strVarIfInTF {lTmpaStr} {a} {prgReturn{Yes}} {prgReturn{No}}
\end{verbatim}

\begin{verbatim}
\strCompare {⟨tl1⟩} {⟨relation⟩} {⟨tl2⟩} {prgReturn{true code}} {prgReturn{false code}}
\strCompareT {⟨tl1⟩} {⟨relation⟩} {⟨tl2⟩} {prgReturn{true code}} {prgReturn{false code}}
\strCompareF {⟨tl1⟩} {⟨relation⟩} {⟨tl2⟩} {prgReturn{true code}} {prgReturn{false code}}
\strCompareTF {⟨tl1⟩} {⟨relation⟩} {⟨tl2⟩} {prgReturn{true code}} {prgReturn{false code}}
\end{verbatim}

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

\begin{verbatim}
\strCompareTF {ab} {⟨abc⟩} {prgReturn{Yes}} {prgReturn{No}}
\strCompareTF {ab} {⟨aa⟩} {prgReturn{Yes}} {prgReturn{No}}
\end{verbatim}

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:
6.9 String Case Functions

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

\IgnoreSpacesOn
\strCase (bbb) {
  {aaa} {\prgReturn{First}}
  {bbb} {\prgReturn{Second}}
  {ccb} {\prgReturn{Third}}
}
\IgnoreSpacesOff

\strCaseT \{⟨test string⟩\}
{⟨string case₁⟩} {⟨code case₁⟩}
{⟨string case₂⟩} {⟨code case₂⟩}
...
{⟨string caseₙ⟩} {⟨code caseₙ⟩}
{⟨true code⟩}

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. If any of the cases are matched, the ⟨true code⟩ is also inserted into the input stream (after the code for the appropriate case).

\IgnoreSpacesOn
\strCaseT {bbb} {
  {aaa} {\tlSet\lTmpkTl{First}}
  {bbb} {\tlSet\lTmpkTl{Second}}
  {ccb} {\tlSet\lTmpkTl{Third}}
}{
  \prgReturn{\tlUse\lTmpkTl}
}
\IgnoreSpacesOff

\strCaseF \{⟨test string⟩\}
{⟨string case₁⟩} {⟨code case₁⟩}
{⟨string case₂⟩} {⟨code case₂⟩}
...
{⟨string caseₙ⟩} {⟨code caseₙ⟩}
{⟨false code⟩}

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 \texttt{\textbackslash strIfEq}) then the associated \langle code\rangle is left in the input stream and other cases are discarded. If none match then the \langle false code\rangle is inserted.

\begin{verbatim}
\IgnoreSpacesOn
\strCaseF \{bbb\} {
  {aaa} {\prgReturn{First}}
  {bbb} {\prgReturn{Second}}
  {ccb} {\prgReturn{Third}}
}\{ 
  \prgReturn{No-Match!}
}
\IgnoreSpacesOff
\end{verbatim}

\begin{verbatim}
\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)\}
\end{verbatim}

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

\begin{verbatim}
\IgnoreSpacesOn
\strCaseTF \{bbb\} {
  {aaa} {\tlSet\lTmpkTl{First}}
  {bbb} {\tlSet\lTmpkTl{Second}}
  {ccb} {\tlSet\lTmpkTl{Third}}
}\{ 
  \prgReturn{\tlUse\lTmpkTl}
}
\IgnoreSpacesOff
\end{verbatim}
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 The Syntax of Integer Expressions

An integer expression should consist, after evaluation of functions defined with \PrgNewFunction and expansion, of +, -, *, /, (, ) and of course integer operands. The result is calculated by applying standard mathematical rules with the following peculiarities:

- / denotes division rounded to the closest integer with ties rounded away from zero;
there is an error and the overall expression evaluates to zero whenever the absolute value of any intermediate result exceeds \(2^{31} - 1\), except in the case of scaling operations \(a*b/c\), for which \(a*b\) may be arbitrarily large (but the operands \(a, b, c\) are still constrained to an absolute value at most \(2^{31} - 1\));

- parentheses may not appear after unary + or -, namely placing +( or -( at the start of an expression or after +, -, *, / or ( leads to an error.

Each integer operand can be either an integer variable (with no need for `\intUse`) or an integer denotation. For example both of the following give the same result because `\lFooSomeTl` expands to the integer denotation 5 while the integer variable `\lFooSomeInt` takes the value 4.

\[
\begin{align*}
\text{\intEval}\{5 + 4 \times 3 - (3 + 4 \times 5)\} & = -6 \\
\text{\tlNew}\ \lFooSomeTl \\
\text{\tlSet}\ \lFooSomeTl\{5\} \\
\text{\intNew}\ \lFooSomeInt \\
\text{\intSet}\ \lFooSomeInt\{4\} \\
\text{\intEval}\{\lFooSomeTl + \lFooSomeInt \times 3 - (3 + 4 \times 5)\} & = -6
\end{align*}
\]

### 7.3 Using Integer Expressions

`\intEval\{\langle\text{integer expression}\rangle\}`

Evaluates the \(\langle\text{integer expression}\rangle\) 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

\[
\begin{align*}
\text{\intEval}\{(1+4)\times(2-3)/5\} & = -1 \\
\text{\intEval}\{\strCount{12\TeX34} - \tlCount{12\TeX34}\} & = 4
\end{align*}
\]

`\intMathAdd\{\langle\text{integer expression}\rangle_1\}\{\langle\text{integer expression}\rangle_2\}`

Adds \(\langle\text{integer expression}\rangle_1\) and \(\langle\text{integer expression}\rangle_2\), and returns the result. For example

\[
\begin{align*}
\text{\intMathAdd}\{7\}\{3\} & = 10
\end{align*}
\]

`\intMathSub\{\langle\text{integer expression}\rangle_1\}\{\langle\text{integer expression}\rangle_2\}`

Subtracts \(\langle\text{integer expression}\rangle_2\) from \(\langle\text{integer expression}\rangle_1\), and returns the result. For example

\[
\begin{align*}
\text{\intMathSub}\{7\}\{3\} & = 4
\end{align*}
\]

`\intMathMult\{\langle\text{integer expression}\rangle_1\}\{\langle\text{integer expression}\rangle_2\}`

Multiplies \(\langle\text{integer expression}\rangle_1\) by \(\langle\text{integer expression}\rangle_2\), and returns the result. For example

\[
\begin{align*}
\text{\intMathMult}\{7\}\{3\} & = 21
\end{align*}
\]
\texttt{\texttt{	extbackslash intMathDiv}} \{\langle \text{integer expression}_1 \rangle\} \{\langle \text{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

\texttt{\texttt{	extbackslash intMathDiv}} \{8\} \{3\}

\texttt{3}

\texttt{\texttt{	extbackslash intMathDivTruncate}} \{\langle \text{integer expression}_1 \rangle\} \{\langle \text{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

\texttt{\texttt{	extbackslash intMathDivTruncate}} \{8\} \{3\}

\texttt{2}

\texttt{\texttt{	extbackslash intMathSign}} \{\langle intexpr \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.

\texttt{\texttt{	extbackslash intMathAbs}} \{\langle integer expression \rangle\}

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

\texttt{\texttt{	extbackslash intMathMax}} \{\langle intexpr_1 \rangle\} \{\langle intexpr_2 \rangle\}
\texttt{\texttt{	extbackslash intMathMin}} \{\langle intexpr_1 \rangle\} \{\langle intexpr_2 \rangle\}

Evaluates the \langle integer expressions \rangle as described for \texttt{\texttt{	extbackslash intEval}} and leaves either the larger or smaller value in the input stream as an \langle integer denotation \rangle after two expansions.

\texttt{\texttt{	extbackslash intMathMod}} \{\langle intexpr_1 \rangle\} \{\langle intexpr_2 \rangle\}

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{\texttt{	extbackslash intMathDivTruncate}} \{\langle intexpr_1 \rangle\} \{\langle intexpr_2 \rangle\} 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{	extbackslash intMathRand}} \{\langle intexpr_1 \rangle\} \{\langle intexpr_2 \rangle\}

Evaluates the two \langle integer expressions \rangle and produces a pseudo-random number between the two (with bounds included).

\textbf{7.4 Creating and Using Integers}

\texttt{\texttt{	extbackslash intNew}} \langle integer \rangle

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.
\intConst\{integer\}\{(integer\ expression)\}

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 \(\langle integer\ expression\rangle\).

\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.

### 7.5 Viewing Integers

\intLog\{(integer\ expression)\}

Writes the result of evaluating the \(\langle integer\ expression\rangle\) in the log file.

\intVarLog\{integer\}

Writes the value of the \(\langle integer\rangle\) in the log file.

\intShow\{(integer\ expression)\}

Displays the result of evaluating the \(\langle integer\ expression\rangle\) on the terminal.

\intVarShow\{integer\}

Displays the value of the \(\langle integer\rangle\) on the terminal.

### 7.6 Setting Integer Variables

\intSet\{integer\}\{(integer\ expression)\}

Sets \(\langle integer\rangle\) to the value of \(\langle integer\ expression\rangle\), which must evaluate to an integer (as described for \intEval). For example

\begin{verbatim}
\intSet \lTmpaInt \{3+5\}
\intUse \lTmpaInt
\end{verbatim}

\intSetEq\{integer\_1\}\{integer\_2\}

Sets the content of \(\langle integer\_1\rangle\) equal to that of \(\langle integer\_2\rangle\).

\intZero\{integer\}

Sets \(\langle integer\rangle\) to 0. For example

\begin{verbatim}
\intSet \lTmpaInt \{5\}
\intZero \lTmpaInt
\intUse \lTmpaInt
\end{verbatim}
\texttt{\textbf{intZeroNew} (integer)}

Ensures that the ⟨integer⟩ exists globally by applying \texttt{\textbf{intNew}} if necessary, then applies \texttt{\textbf{intZero}} to leave the ⟨integer⟩ set to zero.

\texttt{\textbf{intIncr} (integer)}

Increases the value stored in ⟨integer⟩ by 1. For example

\begin{verbatim}
\texttt{intSet \lTmpaInt \{5\}}
\texttt{intIncr \lTmpaInt}
\texttt{intUse \lTmpaInt}
\end{verbatim}

\texttt{\textbf{intDecr} (integer)}

Decreases the value stored in ⟨integer⟩ by 1. For example

\begin{verbatim}
\texttt{intSet \lTmpaInt \{5\}}
\texttt{intDecr \lTmpaInt}
\texttt{intUse \lTmpaInt}
\end{verbatim}

\texttt{\textbf{intAdd} (integer) \{⟨integer expression⟩\}}

Adds the result of the ⟨integer expression⟩ to the current content of the ⟨integer⟩. For example

\begin{verbatim}
\texttt{intSet \lTmpaInt \{5\}}
\texttt{intAdd \lTmpaInt \{2\}}
\texttt{intUse \lTmpaInt}
\end{verbatim}

\texttt{\textbf{intSub} (integer) \{⟨integer expression⟩\}}

Subtracts the result of the ⟨integer expression⟩ from the current content of the ⟨integer⟩. For example

\begin{verbatim}
\texttt{intSet \lTmpaInt \{5\}}
\texttt{intSub \lTmpaInt \{3\}}
\texttt{intUse \lTmpaInt}
\end{verbatim}

\section{7.7 Integer Step Functions}

\texttt{\textbf{intReplicate} \{⟨integer expression⟩\} \{⟨tokens⟩\}}

Evaluates the ⟨integer expression⟩ (which should be zero or positive) and returns the resulting number of copies of the ⟨tokens⟩.

\begin{verbatim}
\texttt{intReplicate \{4\} \{Hello\}}
\end{verbatim}

\texttt{\textbf{intStepInline} \{⟨initial value⟩\} \{⟨step⟩\} \{⟨final value⟩\} \{⟨code⟩\}}

This function first evaluates the ⟨initial value⟩, ⟨step⟩ and ⟨final value⟩, all of which should be integer expressions. Then for each ⟨value⟩ from the ⟨initial value⟩ to the ⟨final value⟩ in turn (using ⟨step⟩ 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).

\IgnoreSpacesOn
\tlClear \lTmpaTl
\intStepInline {1} {3} {30} {
    \tlPutRight \lTmpaTl {{\#1}}
}
\tlUse \lTmpaTl
\IgnoreSpacesOff

\intStepOneInline {\langle initial value \rangle} {\langle final value \rangle} \langle code \rangle

This function first evaluates the \langle initial value \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 a fixed step of 1 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).

\IgnoreSpacesOn
\tlClear \lTmpaTl
\intStepOneInline {1} {10} {
    \tlPutRight \lTmpaTl {{\#1}}
}
\tlUse \lTmpaTl
\IgnoreSpacesOff

\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.

\intStepOneVariable {\langle initial value \rangle} {\langle final value \rangle} \langle tl var \rangle \langle code \rangle

This function first evaluates the \langle initial value \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 a fixed step of 1 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.

7.8 Integer Conditionals

\intIfExist \langle integer \rangle
\intIfExistT \langle integer \rangle \langle true code \rangle
\intIfExistF \langle integer \rangle \langle false code \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\text{integer expression}\rangle\} \\
\intIfOddT \{\langle\text{integer expression}\rangle\} \{\langle\text{true code}\rangle\} \\
\intIfOddF \{\langle\text{integer expression}\rangle\} \{\langle\text{false code}\rangle\} \\
\intIfOddTF \{\langle\text{integer expression}\rangle\} \{\langle\text{true code}\rangle\} \{\langle\text{false code}\rangle\}

This function first evaluates the \(\text{integer expression}\) as described for \texttt{\intEval}. It then evaluates if this is odd or even, as appropriate.

\intIfEven \{\langle\text{integer expression}\rangle\} \\
\intIfEvenT \{\langle\text{integer expression}\rangle\} \{\langle\text{true code}\rangle\} \\
\intIfEvenF \{\langle\text{integer expression}\rangle\} \{\langle\text{false code}\rangle\} \\
\intIfEvenTF \{\langle\text{integer expression}\rangle\} \{\langle\text{true code}\rangle\} \{\langle\text{false code}\rangle\}

This function first evaluates the \(\text{integer expression}\) as described for \texttt{\intEval}. It then evaluates if this is even or odd, as appropriate.

\intCompare \{\langle\text{intexpr}_1\rangle\} \{\langle\text{relation}\rangle\} \{\langle\text{intexpr}_2\rangle\} \\
\intCompareT \{\langle\text{intexpr}_1\rangle\} \{\langle\text{relation}\rangle\} \{\langle\text{intexpr}_2\rangle\} \{\langle\text{true code}\rangle\} \\
\intCompareF \{\langle\text{intexpr}_1\rangle\} \{\langle\text{relation}\rangle\} \{\langle\text{intexpr}_2\rangle\} \{\langlefalse code\rangle\} \\
\intCompareTF \{\langle\text{intexpr}_1\rangle\} \{\langle\text{relation}\rangle\} \{\langle\text{intexpr}_2\rangle\} \{\langle\text{true code}\rangle\} \{\langle\text{false code}\rangle\}

This function first evaluates each of the \(\text{integer expressions}\) as described for \texttt{\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

\begin{verbatim}
\intCompareTF \{2\} > \{1\} \{\texttt{\progReturn{Greater}}\} \{\texttt{\progReturn{Less}}\}
\intCompareTF \{2\} > \{3\} \{\texttt{\progReturn{Greater}}\} \{\texttt{\progReturn{Less}}\}
\end{verbatim}  

\textbf{Greater} \textbf{Less}

\section{7.9 Integer Case Functions}

\intCase \{\langle\text{test integer expression}\rangle\} \\
\{ \\
\{\langle\text{intexpr case}_1\rangle\} \{\langle\text{code case}_1\rangle\} \\
\{\langle\text{intexpr case}_2\rangle\} \{\langle\text{code case}_2\rangle\} \\
\ldots \\
\{\langle\text{intexpr case}_n\rangle\} \{\langle\text{code case}_n\rangle\}
\}

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 \{\langle\text{test integer expression}\rangle\} \\
\{ \\
\{\langle\text{intexpr case}_1\rangle\} \{\langle\text{code case}_1\rangle\} \\
\{\langle\text{intexpr case}_2\rangle\} \{\langle\text{code case}_2\rangle\} \\
\ldots \\
\{\langle\text{intexpr case}_n\rangle\} \{\langle\text{code case}_n\rangle\}
\} \\
\{\langle\text{true code}\rangle\}

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 (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).

\intCaseF \{(test integer expression)\}
\{  
  \{(intexpr case_1)\} \{(code case_1)\}
  \{(intexpr case_2)\} \{(code case_2)\}
  \ldots
  \{(intexpr case_n)\} \{(code case_n)\}
\}
\{(false code)\}

This function evaluates the (test integer expression) and compares this in turn to each of the (integer expression cases). If the two are equal then the associated (code) is left in the input stream and other cases are discarded. If none match then the (false code) is into the input stream (after the code for the appropriate case). For example

\IgnoreSpacesOn
\intCaseF \{ 2 * 5 \}
\{  
  \{ 5 \} \{ Small \}
  \{ 4 + 6 \} \{ Medium \}
  \{ -2 * 10 \} \{ Negative \}
\}
\{ No idea! \}
\IgnoreSpacesOff

\intCaseTF \{(test integer expression)\}
\{  
  \{(intexpr case_1)\} \{(code case_1)\}
  \{(intexpr case_2)\} \{(code case_2)\}
  \ldots
  \{(intexpr case_n)\} \{(code case_n)\}
\}
\{(true code)\}
\{(false code)\}

This function evaluates the (test integer expression) and compares this in turn to each of the (integer 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), while if none match then the (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° 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 The Syntax of Floating Point Expressions

A decimal floating point number is one which is stored as a significand and a separate exponent. The module implements expandably a wide set of arithmetic, trigonometric, and other operations on decimal floating point numbers, to be used within floating point expressions. Floating point expressions support the following operations with their usual precedence.

- **Basic arithmetic:** addition \( x + y \), subtraction \( x - y \), multiplication \( x \times y \), division \( x/y \), square root \( \sqrt{x} \), and parentheses.
- **Comparison operators:** \( x < y \), \( x \leq y \), \( x > y \), \( x \geq y \) etc.
- **Boolean logic:** sign sign \( x \), negation \( !x \), conjunction \( x \& \& y \), disjunction \( x \| y \), ternary operator \( x ? y : z \).
- **Exponentials:** \( \exp x \), \( \ln x \), \( x^y \), \( \log_b x \).
- **Integer factorial:** \( \text{fact} x \).
- **Trigonometry:** \( \sin x \), \( \cos x \), \( \tan x \), \( \cot x \), \( \sec x \), \( \csc x \) expecting their arguments in radians, and \( \sin_d x \), \( \cos_d x \), \( \tan_d x \), \( \cot_d x \), \( \sec_d x \), \( \csc_d x \) expecting their arguments in degrees.
- **Inverse trigonometric functions:** \( \arcsin x \), \( \arccos x \), \( \arctan x \), \( \arccot x \), \( \arcsec x \), \( \arccsc x \) giving a result in radians, and \( \arcsin_d x \), \( \arccos_d x \), \( \arctan_d x \), \( \arccot_d x \), \( \arcsec_d x \), \( \arccsc_d x \) giving a result in degrees.
- **Extrema:** \( \max(x_1, x_2, \ldots) \), \( \min(x_1, x_2, \ldots) \), \( \text{abs}(x) \).
- **Rounding functions**, controlled by two optional values, \( n \) (number of places, 0 by default) and \( t \) (behavior on a tie, NaN by default):
  - \( \text{trunc}(x, n) \) rounds towards zero,
  - \( \text{floor}(x, n) \) rounds towards \(-\infty\),
  - \( \text{ceil}(x, n) \) rounds towards \(+\infty\),
  - \( \text{round}(x, n, t) \) rounds to the closest value, with ties rounded to an even value by default, towards zero if \( t = 0 \), towards \(+\infty\) if \( t > 0 \) and towards \(-\infty\) if \( t < 0 \).
- **Random numbers:** \( \text{rand}() \), \( \text{randint}(m, n) \).
- **Constants:** \( \pi \), \( \text{deg} \) (one degree in radians).
- **Dimensions**, automatically expressed in points, e.g., \( \text{pc} \) is 12.
- **Automatic conversion** (no need for \( \text{intUse} \), etc) of integer, dimension, and skip variables to floating point numbers, expressing dimensions in points and ignoring the stretch and shrink components of skips.
- **Tuples:** \( (x_1, \ldots, x_n) \) that can be stored in variables, added together, multiplied or divided by a floating point number, and nested.

Floating point numbers can be given either explicitly (in a form such as \( 1.234e-34 \), or \( -.0001 \)), or as a stored floating point variable, which is automatically replaced by its current value. A “floating point” is a floating point number or a tuple thereof.

An example of use could be the following.

\[
\text{\LaTeX{} can now compute: } \frac{\sin(3.5)}{2} + 2\cdot 10^{-3} = \text{fpEval } \{\sin(3.5)/2 + 2e-3\}.
\]

\[
\text{\LaTeX{} can now compute: } \frac{\sin(3.5)}{2} + 2\cdot 10^{-3} = -0.173391613848099.
\]

The operation \( \text{round} \) can be used to limit the result’s precision. Adding \( +0 \) avoids the possibly undesirable output \( -0 \), replacing it by \( +0 \).
8. Using Floating Point Expressions

**\fpEval{\langle floating point expression\rangle}**

Evaluates the \langle floating point expression\rangle 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 \fpEval and they are combined as \(\langle fp_1, fp_2, \ldots, fp_n\rangle\) if \(n > 1\) and \(\langle fp_1\rangle\) or () for fewer items. For example

\[
\begin{align*}
\fpEval{(1.2+3.4)\ast(5.6-7.8)/9} & \quad -1.124444444444444 \\
\fpMathAdd{\langle fpexpr_1\rangle}{\langle fpexpr_2\rangle} & \\
\fpMathSub{\langle fpexpr_1\rangle}{\langle fpexpr_2\rangle} & \\
\fpMathMult{\langle fpexpr_1\rangle}{\langle fpexpr_2\rangle} & \\
\fpMathDiv{\langle fpexpr_1\rangle}{\langle fpexpr_2\rangle} & \\
\fpMathSign{\langle fpexpr\rangle} & 
\end{align*}
\]

**\fpMathAdd{\langle fpexpr_1\rangle}{\langle fpexpr_2\rangle}**

Adds \(\langle fpexpr_1\rangle\) and \(\langle fpexpr_2\rangle\), and returns the result. For example

\[
\begin{align*}
\fpMathAdd{2.8}{3.7} & \quad 6.5 \\
\fpMathAdd{3.8-1}{2.7+1} & \\
\end{align*}
\]

**\fpMathSub{\langle fpexpr_1\rangle}{\langle fpexpr_2\rangle}**

Subtracts \(\langle fpexpr_2\rangle\) from \(\langle fpexpr_1\rangle\), and returns the result. For example

\[
\begin{align*}
\fpMathSub{2.8}{3.7} & \quad -0.9 \\
\fpMathSub{3.8-1}{2.7+1} & \\
\end{align*}
\]

**\fpMathMult{\langle fpexpr_1\rangle}{\langle fpexpr_2\rangle}**

Multiplies \(\langle fpexpr_1\rangle\) by \(\langle fpexpr_2\rangle\), and returns the result. For example

\[
\begin{align*}
\fpMathMult{2.8}{3.7} & \quad 10.36 \\
\fpMathMult{3.8-1}{2.7+1} & \\
\end{align*}
\]

**\fpMathDiv{\langle fpexpr_1\rangle}{\langle fpexpr_2\rangle}**

Divides \(\langle fpexpr_1\rangle\) by \(\langle fpexpr_2\rangle\), and returns the result. For example

\[
\begin{align*}
\fpMathDiv{2.8}{3.7} & \quad 0.7567567567567568 \\
\fpMathDiv{3.8-1}{2.7+1} & \\
\end{align*}
\]

**\fpMathSign{\langle fpexpr\rangle}**

Evaluates the \langle fpexpr\rangle and returns the value using \fpEval{\text{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

\[
\begin{align*}
\fpMathSign{3.5} & \quad 1 \\
\fpMathSign{-2.7} & \\
\end{align*}
\]
\texttt{\textbackslash fpMathAbs \{\textlangle floating point expression\textrangle\}} \hspace{1cm} Evaluates the \langle floating point expression\rangle as described for \texttt{\textbackslash fpEval} and returns the absolute value. If the argument is ±\infty, NaN or a tuple, “invalid operation” occurs. Within floating point expressions, \texttt{abs()} can be used; it accepts ±\infty and NaN as arguments.

\texttt{\textbackslash fpMathMax \{\langle fp expression\rangle_1\} \{\langle fp expression\rangle_2\}} \hspace{1cm} \texttt{\textbackslash fpMathMin \{\langle fp expression\rangle_1\} \{\langle fp expression\rangle_2\}} \hspace{1cm} Evaluates the \langle floating point expressions\rangle as described for \texttt{\textbackslash fpEval} and returns the resulting larger (max) or smaller (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.

### 8.4 Creating and Using Floating Points

\texttt{\textbackslash fpNew \{fp var\}} \hspace{1cm} Creates a new \langle fp var\rangle or raises an error if the name is already taken. The declaration is global. The \langle fp var\rangle is initially +0.

\texttt{\textbackslash fpConst \{fp var\} \{\langle floating point expression\rangle\}} \hspace{1cm} Creates a new constant \langle fp var\rangle or raises an error if the name is already taken. The \langle fp var\rangle is set globally equal to the result of evaluating the \langle floating point expression\rangle. For example

\begin{verbatim}
\fpConst \cMyPiFp {3.1415926}
\fpUse \cMyPiFp 3.1415926
\end{verbatim}

\texttt{\textbackslash fpUse \{fp var\}} \hspace{1cm} Recovers the value of the \langle fp var\rangle and returns the value as a decimal number with no exponent.

### 8.5 Viewing Floating Points

\texttt{\textbackslash fpLog \{\langle floating point expression\rangle\}} \hspace{1cm} Evaluates the \langle floating point expression\rangle and writes the result in the log file.

\texttt{\textbackslash fpVarLog \{fp var\}} \hspace{1cm} Writes the value of \langle fp var\rangle in the log file.

\texttt{\textbackslash fpShow \{\langle floating point expression\rangle\}} \hspace{1cm} Evaluates the \langle floating point expression\rangle and displays the result in the terminal.

\texttt{\textbackslash fpVarShow \{fp var\}} \hspace{1cm} Displays the value of \langle fp var\rangle in the terminal.
8.6 Setting Floating Point Variables

\texttt{\textbackslash fpSet} \texttt{} \texttt{(fp var)} \texttt{\{}\texttt{\langle floating point expression\rangle}\texttt{}}

Sets \texttt{(fp var)} equal to the result of computing the \texttt{(floating point expression)}. For example

\begin{verbatim}
\fpSet \lTmpaFp {4/7}
\fpUse \lTmpaFp
\end{verbatim}

\begin{verbatim}
0.5714285714285714
\end{verbatim}

\texttt{\textbackslash fpSetEq} \texttt{} \texttt{(fp var\textsubscript{1})} \texttt{\{}\texttt{\langle fp var\textsubscript{2}\rangle}\texttt{}}

Sets the floating point variable \texttt{(fp var\textsubscript{1})} equal to the current value of \texttt{(fp var\textsubscript{2})}.

\texttt{\textbackslash fpZero} \texttt{} \texttt{(fp var)}

Sets the \texttt{(fp)} to +0. For example

\begin{verbatim}
\fpSet \lTmpaFp {5.3}
\fpZero \lTmpaFp
\fpUse \lTmpaFp
\end{verbatim}

\begin{verbatim}
0
\end{verbatim}

\texttt{\textbackslash fpZeroNew} \texttt{} \texttt{(fp var)}

Ensures that the \texttt{(fp var)} exists globally by applying \texttt{\textbackslash fpNew} if necessary, then applies \texttt{\textbackslash fpZero} to leave the \texttt{(fp var)} set to +0.

\texttt{\textbackslash fpAdd} \texttt{} \texttt{(fp var)} \texttt{\{}\texttt{\langle floating point expression\rangle}\texttt{}}

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

\begin{verbatim}
\fpSet \lTmpaFp {5.3}
\fpAdd \lTmpaFp {2.11}
\fpUse \lTmpaFp
\end{verbatim}

\begin{verbatim}
7.41
\end{verbatim}

\texttt{\textbackslash fpSub} \texttt{} \texttt{(fp var)} \texttt{\{}\texttt{\langle floating point expression\rangle}\texttt{}}

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

\begin{verbatim}
\fpSet \lTmpaFp {5.3}
\fpSub \lTmpaFp {2.11}
\fpUse \lTmpaFp
\end{verbatim}

\begin{verbatim}
3.19
\end{verbatim}

8.7 Floating Point Step Functions

\texttt{\textbackslash fpStepInline} \texttt{} \texttt{\{}\texttt{(initial value)}\texttt{\}} \texttt{\{}\texttt{(step)}\texttt{\}} \texttt{\{}\texttt{(final value)}\texttt{\}} \texttt{\{}\texttt{(code)}\texttt{\}}

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

```
\IgnoreSpacesOn
\tlClear \lTmpaTl
\fpStepInline {1} {0.1} {1.5} {
  \tlPutRight \lTmpaTl {[#1]}
}
\tlUse \lTmpaTl
\IgnoreSpacesOff
```

This function first evaluates the ⟨initial value⟩, ⟨step⟩ and ⟨final value⟩, all of which should be floating point expressions evaluating to a floating point number, not a tuple. Then for each ⟨value⟩ from the ⟨initial value⟩ to the ⟨final value⟩ in turn (using ⟨step⟩ between each ⟨value⟩), the ⟨code⟩ is inserted into the input stream, with the ⟨tl var⟩ defined as the current ⟨value⟩. Thus the ⟨code⟩ should make use of the ⟨tl var⟩.

### 8.8 Float Point Conditionals

```
\fpIfExist {\fp var} \fpIfExistT {\fp var} {⟨true code⟩} \fpIfExistF {\fp var} {⟨false code⟩} \fpIfExistTF {\fp var} {⟨true code⟩} {⟨false code⟩}
```

Tests whether the ⟨fp var⟩ is currently defined. This does not check that the ⟨fp var⟩ really is a floating point variable. For example

```
\fpIfExistTF \lTmpaFp {\prgReturn{Yes}} {\prgReturn{No}}
\fpIfExistTF \lMyUndefinedFp {\prgReturn{Yes}} {\prgReturn{No}}
```

```
\fpCompare {⟨fpexpr1⟩} ⟨relation⟩ {⟨fpexpr2⟩}
\fpCompareT {⟨fpexpr1⟩} ⟨relation⟩ {⟨fpexpr2⟩} {⟨true code⟩}
\fpCompareF {⟨fpexpr1⟩} ⟨relation⟩ {⟨fpexpr2⟩} {⟨false code⟩}
\fpCompareTF {⟨fpexpr1⟩} ⟨relation⟩ {⟨fpexpr2⟩} {⟨true code⟩} {⟨false code⟩}
```

Compares the ⟨fpexpr1⟩ and the ⟨fpexpr2⟩, and returns true if the ⟨relation⟩ is obeyed. For example

```
\fpCompareTF {1} > {0.9999} {\prgReturn{Greater}} {\prgReturn{Less}}
\fpCompareTF {1} > {1.0001} {\prgReturn{Greater}} {\prgReturn{Less}}
```

Two floating points \(x\) and \(y\) may obey four mutually exclusive relations: \(x < y\), \(x = y\), \(x > y\), or \(?\)?y (“not ordered”). The last case occurs exactly if one or both operands is NaN or is a tuple, unless they are equal tuples. Note that a NaN is distinct from any value, even another NaN, hence \(x = x\) 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). At present any other comparison with tuples yields ? (not ordered). This is experimental.
Chapter 9

Dimensions (\texttt{Dim})

9.1 Constant and Scratch Dimensions

\texttt{\textbackslash cMaxDim}

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

\texttt{\textbackslash cZeroDim}

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

\texttt{\textbackslash lTmpaDim \textbackslash lTmpbDim \textbackslash lTmpcDim \textbackslash lTmpiDim \textbackslash lTmpjDim \textbackslash lTmpkDim}

Scratch dimensions for local assignment. These are never used by the \texttt{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.

\texttt{\textbackslash gTmpaDim \textbackslash gTmpbDim \textbackslash gTmpcDim \textbackslash gTmpiDim \textbackslash gTmpjDim \textbackslash gTmpkDim}

Scratch dimensions for global assignment. These are never used by the \texttt{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

\texttt{\textbackslash 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 \texttt{\textbackslash dimUse/\textbackslash tlUse}) and applying the standard mathematical rules. The result of the calculation is returned as a \langle dimension denotation\rangle. For example

\begin{verbatim}
\dimEval \{(1.2pt+3.4pt)/9\} 0.51111pt
\end{verbatim}

\texttt{\textbackslash 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

\begin{verbatim}
\dimMathAdd \{2.8pt\} \{3.7pt\} 6.5pt
\dimMathAdd \{3.8pt-1pt\} \{2.7pt+1pt\} 6.5pt
\end{verbatim}
\dimMathSub \{\dimexpr 1\}\{\dimexpr 2\}

Subtracts \{\dimexpr 2\} from \{\dimexpr 1\}, and returns the result. For example

\dimMathSub {2.8pt} {3.7pt}
\dimMathSub {3.8pt-1pt} {2.7pt+1pt}

\dimMathRatio \{\dimexpr 1\}\{\dimexpr 2\}

Parses the two \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

\dimMathRatio {5pt} {10pt}

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

\dimSet \lTmpaDim {10pt*\dimMathRatio\{5pt\}\{10pt\}}

\dimMathSign \{\dimexpr\}

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

\dimMathSign {3.5pt}
\dimMathSign {-2.7pt}

\dimMathAbs \{\dimexpr\}

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

\dimMathAbs {3.5pt}
\dimMathAbs {-2.7pt}

\dimMathMax \{\dimexpr 1\}\{\dimexpr 2\}
\dimMathMin \{\dimexpr 1\}\{\dimexpr 2\}

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

\dimMathMax {3.5pt} {-2.7pt}
\dimMathMin {3.5pt} {-2.7pt}

9.3 Creating and Using Dimensions

\dimNew \langle dimension\rangle

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

\texttt{\textbackslash dimConst \{\texttt{dimension} \}\{\texttt{dimension expression}\}}

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{\textbackslash dimUse \{\texttt{dimension}\}}

Recovers 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.

### 9.4 Viewing Dimensions

\texttt{\textbackslash 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{\textbackslash dimVarLog \{\texttt{dimension}\}}

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

\begin{verbatim}
\dimVarLog \lFooSomeDim
\end{verbatim}

\texttt{\textbackslash 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{\textbackslash dimVarShow \{\texttt{dimension}\}}

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

\begin{verbatim}
\dimVarShow \lFooSomeDim
\end{verbatim}

### 9.5 Setting Dimension Variables

\texttt{\textbackslash dimSet \{\texttt{dimension}\} \{\texttt{dimension expression}\}}

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

\texttt{\textbackslash dimSetEq \{\texttt{dimension}_1\} \{\texttt{dimension}_2\}}

Sets the content of \texttt{(dimension}_1\) equal to that of \texttt{(dimension}_2\). For example
\dimSet \lTmaaDim \{10pt\}
\dimSetEq \lTmbbDim \lTmaaDim
\dimUse \lTmbbDim

\dimZero \langle dimension \rangle

Sets \langle dimension \rangle to 0 pt. For example

\dimSet \lTmaaDim \{1em\}
\dimZero \lTmaaDim
\dimUse \lTmaaDim

\dimZeroNew \langle dimension \rangle

Ensures that the \langle dimension \rangle exists globally by applying \dimNew if necessary, then applies \dimZero to set the \langle dimension \rangle to zero. For example

\dimZeroNew \lFooSomeDim
\dimUse \lFooSomeDim

\dimAdd \langle dimension \rangle \{\langle dimension expression \rangle\}

Adds the result of the \langle dimension expression \rangle to the current content of the \langle dimension \rangle. For example

\dimSet \lTmaaDim \{5.3pt\}
\dimAdd \lTmaaDim \{2.11pt\}
\dimUse \lTmaaDim

\dimSub \langle dimension \rangle \{\langle dimension expression \rangle\}

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

\dimSet \lTmaaDim \{5.3pt\}
\dimSub \lTmaaDim \{2.11pt\}
\dimUse \lTmaaDim

\dimStepInline \{\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 dimension 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).

9.6 Dimension Step Functions
\IgnoreSpacesOn
\tlClear \lTmpaTl
\dimStepInline {1pt} {0.1pt} {1.5pt} {
  \tlPutRight \lTmpaTl {[#1]}
}
\tlUse \lTmpaTl
\IgnoreSpacesOff
[1.0pt][1.1pt][1.20001pt][1.30002pt][1.40002pt]

\dimStepVariable \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 dimension 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 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.

9.7 Dimension Conditionals

\dimIfExist \langle dimension \rangle
\dimIfExistT \langle dimension \rangle \langle true code \rangle
\dimIfExistF \langle dimension \rangle \langle false code \rangle
\dimIfExistTF \langle dimension \rangle \langle true code \rangle \langle false code \rangle

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

\dimIfExistTF \lTmpaDim \prgReturn{Yes} \prgReturn{No}
\dimIfExistTF \lFooUndefinedDim \prgReturn{Yes} \prgReturn{No}

\dimCompare \langle dimexpr1 \rangle \langle relation \rangle \langle dimexpr2 \rangle
\dimCompareT \langle dimexpr1 \rangle \langle relation \rangle \langle dimexpr2 \rangle \langle true code \rangle
\dimCompareF \langle dimexpr1 \rangle \langle relation \rangle \langle dimexpr2 \rangle \langle false code \rangle
\dimCompareTF \langle dimexpr1 \rangle \langle relation \rangle \langle dimexpr2 \rangle \langle true code \rangle \langle false code \rangle

This function first evaluates each of the \langle dimension expressions \rangle as described for \dimEval. The two results are then compared using the \langle relation \rangle:

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

For example

\dimCompareTF {1pt} > {0.9999pt} \prgReturn{Greater} \prgReturn{Less}
\dimCompareTF {1pt} > {1.0001pt} \prgReturn{Greater} \prgReturn{Less}

Greater Less
9.8 Dimension Case Functions

\dimCase {⟨test dimension expression⟩}
{
  {⟨dimexpr case_1⟩} {⟨code case_1⟩}
  {⟨dimexpr case_2⟩} {⟨code case_2⟩}
  ...
  {⟨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 case_1⟩} {⟨code case_1⟩}
  {⟨dimexpr case_2⟩} {⟨code case_2⟩}
  ...
  {⟨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 case_1⟩} {⟨code case_1⟩}
  {⟨dimexpr case_2⟩} {⟨code case_2⟩}
  ...
  {⟨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

\GrabTable
\dimSet \lTmpaDim {5pt}
\dimCaseF {2\lTmpaDim} {\
  {5pt} \prgReturn{Small}\
  {4pt+6pt} \prgReturn{Medium}\
  {-10pt} \prgReturn{Negative}\
  {\prgReturn {No Match}}\
}
\IgnoreSpacesOff

\medium
\texttt{dimCaseTF \{test dimension expression\} }

\hspace{1cm} \{ \langle \text{dimexpr case}_1 \rangle \} \{ \langle \text{code case}_1 \rangle \} \\
\hspace{1cm} \{ \langle \text{dimexpr case}_2 \rangle \} \{ \langle \text{code case}_2 \rangle \} \\
\hspace{1cm} \ldots \cr 
\hspace{1cm} \{ \langle \text{dimexpr case}_n \rangle \} \{ \langle \text{code case}_n \rangle \} \\
\hspace{1cm} \{ \langle \text{true code} \rangle \} \\
\hspace{1cm} \{ \langle \text{false code} \rangle \} \\

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.

\clistNew \lFooSomeClist

\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).

\clistConst \cFooSomeClist {one,two,three}

\clistVarJoin (clist var) {⟨separator⟩}

Returns the contents of the (clist var), with the (separator) between the items.
10.3 Viewing Comma Lists

\clistLog \{\langle tokens\rangle\}

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

\clistLog \{one,two,three\}

\clistVarLog \{comma list\}

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

\clistSet \lTmpaClist \{a , b , , c , {de} , f \}
\clistVarJoinExtended \lTmpaClist \{ and \} \{, \} \{, \} \{, \} \{, \} \{, \} \{, and \}
a, b, c, de, and f

\clistJoinExtended \{a , b \} \{ and \} \{, \} \{, \} \{, \} \{, \} \{, and \}
a and b

\clistJoinExtended \{a , b , , c , {de} , f \} \{ and \} \{, \} \{, \} \{, \} \{, \} \{, and \}
a, b, c, de, and f

\clistLog \{one,two,three\}

\clistVarLog \lTmpaClist
\clistShow \{\langle tokens\rangle\}

Displays the entries in the comma list in the terminal.
\clistShow\{one,two,three\}

\clistVarShow⟨comma list⟩Displays the entries in the ⟨comma list⟩ in the terminal.
\clistSet\lTmpaClist\{one,two,three\}
\clistVarShow\lTmpaClist

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⟩\}\}.
\clistSet\lTmpaClist\{one,two,three\}
\clistSetJoin\lTmpaClist\{and\}
\clistSetEq⟨comma list₁⟩⟨comma list₂⟩Sets the content of ⟨comma list₁⟩ equal to that of ⟨comma list₂⟩. 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.
\clistSet\lTmpaClist\{one,two,three,four\}
\clistSetEq\lTmpbClist\lTmpaClist
\clistSetJoin\lTmpbClist\{and\}
\clistSetFromSeq⟨comma list⟩⟨sequence⟩Converts the data in the ⟨sequence⟩ into a ⟨comma list⟩: the original ⟨sequence⟩ is unchanged. Items which contain either spaces or commas are surrounded by braces.
\seqPutRight\lTmpaSeq\{one\}
\seqPutRight\lTmpaSeq\{two\}
\clistSetFromSeq\lTmpaClist\lTmpaSeq
\clistSetJoin\lTmpaClist\{and\}
\clistClear⟨comma list⟩Clears all items from the ⟨comma list⟩.
\clistSet\lTmpaClist\{one,two,three,four\}
\clistClear\lTmpaClist
\clistClearNew \lFooSomeClist
\clistSet \lFooSomeClist {one,two,three}
\clistVarJoin \lFooSomeClist { and }

\clistConcat (comma list_1) (comma list_2) (comma list_3)

Concatenates the content of (comma list_2) and (comma list_3) together and saves the result in (comma list_1). The items in (comma list_2) are placed at the left side of the new comma list.

\clistPutLeft (comma list) {{item_1},…,{{item_n}}}

Appends the (items) to the left of the (comma list). 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 (tokens) as a single (item) even if the (tokens) contain commas or spaces, add a set of braces: \clistPutLeft (comma list) { {{\langle tokens\rangle}} }.

\clistPutRight (comma list) {{item_1},…,{{item_n}}}

Appends the (items) to the right of the (comma list). 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 (tokens) as a single (item) even if the (tokens) contain commas or spaces, add a set of braces: \clistPutRight (comma list) { {{\langle 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.

\clistVarRemoveDuplicates (comma list)

Removes duplicate items from the (comma list), leaving the left most copy of each item in the (comma list). The (item) comparison takes place on a token basis, as for \tlIfEqTF.
\clistSet \lTmpaClist {one,two,one,two,three}
\clistVarRemoveDuplicates \lTmpaClist
\clistVarJoin \lTmpaClist {,}

\clistVarRemoveAll ⟨comma list⟩ ⟨(item)⟩
Removes every occurrence of ⟨item⟩ from the ⟨comma list⟩. The ⟨item⟩ comparison takes place on a token basis, as for \tlIfEqTF.
\clistSet \lTmpaClist {one,two,one,two,three}
\clistVarRemoveAll \lTmpaClist {two}
\clistVarJoin \lTmpaClist {,}

\clistVarReverse ⟨comma list⟩
Reverses the order of items stored in the ⟨comma list⟩.
\clistSet \lTmpaClist {one,two,one,two,three}
\clistVarReverse \lTmpaClist
\clistVarJoin \lTmpaClist {,}

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.
\clistSet \lTmpaClist {one,two,three,four}
\clistVarCount \lTmpaClist 4

\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 \clistVarCount) then the function returns nothing.
\tlSet \lTmpaTl {\clistItem {one,two,three,four} {3}}
\tlUse \lTmpaTl

\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.
\clistSet \lTmpaClist {one,two,three,four}
\tlSet \lTmpaTl \clistVarItem \lTmpaClist {3}
\tlUse \lTmpaTl

three

\clistRandItem \langle comma list \rangle
\clistVarRandItem \langle clist var \rangle

Selects a pseudo-random item of the \langle comma list \rangle. If the \langle comma list \rangle has no item, the result is empty.

\tlSet \lTmpaTl {\clistRandItem {one,two,three,four,five,six}}
\tlUse \lTmpaTl
\tlSet \lTmpaTl {\clistRandItem {one,two,three,four,five,six}}
\tlUse \lTmpaTl

six six

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 \langle comma list \rangle \langle token list variable \rangle

Stores the left-most item from the \langle comma list \rangle in the \langle token list variable \rangle without removing it from the \langle comma list \rangle. The \langle token list variable \rangle is assigned locally. If the \langle comma list \rangle is empty the \langle token list variable \rangle is set to the marker value \qNoValue.

\clistSet \lTmpaClist {two,three,four}
\clistGet \lTmpaClist \lTmpaTl
\tlUse \lTmpaTl
two

\clistGetT \langle comma list \rangle \langle token list variable \rangle \langle true code \rangle
\clistGetF \langle comma list \rangle \langle token list variable \rangle \langle false code \rangle
\clistGetTF \langle comma list \rangle \langle token list variable \rangle \langle true code \rangle \langle false code \rangle

If the \langle comma list \rangle is empty, leaves the \langle false code \rangle in the input stream. The value of the \langle token list variable \rangle is not defined in this case and should not be relied upon. If the \langle comma list \rangle is non-empty, stores the left-most item from the \langle comma list \rangle in the \langle token list variable \rangle without removing it from the \langle comma list \rangle. The \langle token list variable \rangle is assigned locally.

\clistSet \lTmpaClist {two,three,four}
\clistGetTF \lTmpaClist \lTmpaTl {\prgReturn{Yes}} {\prgReturn{No}}

Yes

\clistPop \langle comma list \rangle \langle token list variable \rangle

Pops the left-most item from a \langle comma list \rangle into the \langle token list variable \rangle, i.e., removes the item from the comma list and stores it in the \langle token list variable \rangle. The assignment of the \langle token list variable \rangle is local. If the \langle comma list \rangle is empty the \langle token list variable \rangle is set to the marker value \qNoValue.

\clistSet \lTmpaClist {two,three,four}
\clistPop \lTmpaClist \lTmpaTl
\clistVarJoin \lTmpaClist {,}

three,four
CHAPTER 10. COMMA SEPARATED LISTS (CLIST)

\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⟩}

If the ⟨comma list⟩ is empty, leaves the ⟨false code⟩ in the input stream. The value of the ⟨token list variable⟩ is not defined in this case and should not be relied upon. If the ⟨comma list⟩ is non-empty, pops the top item from the ⟨comma list⟩ in the ⟨token list variable⟩, i.e., removes the item from the ⟨comma list⟩. The ⟨token list variable⟩ is assigned locally.

\clistSet \lTmpaClist {two,three,four}
\clistPopTF \lTmpaClist \lTmpaTl {\prgReturn{Yes}} {\prgReturn{No}}
Yes

\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.

\clistSet \lTmpaClist {two,three,four}
\clistPush \lTmpaClist {zero,one}
\clistVarJoin \lTmpaClist |}
zero|one|two|three|four

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\}, 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.

\IgnoreSpacesOn
\tlClear \lTmpaTl
\clistMapInline {one,two,three} {
    \tlPutRight \lTmpaTl {{#1}}
}
\tlUse \lTmpaTl
\IgnoreSpacesOff

\IgnoreSpacesOn
\tlClear \lTmpaTl
\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.
\IgnoreSpacesOn
\clistMapVariable {one,two,three} \lTmpiTl {
  \tlPutRight \gTmpaTl {\expWhole {\lTmpiTl}}}
\tlUse \gTmpaTl
\IgnoreSpacesOff

(one)(two)(three)

10.9 Comma List Conditionals

\clistIfExist ⟨comma list⟩
\clistIfExistT ⟨comma list⟩ {⟨true code⟩}
\clistIfExistF ⟨comma list⟩ {⟨false code⟩}
\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.

\clistIfExistTF \lTmpaClist {\prgReturn{Yes}} {\prgReturn{No}}
\clistIfExistTF \lFooUndefinedClist {\prgReturn{Yes}} {\prgReturn{No}} Yes No

\clistIfEmpty ⟨comma list⟩
\clistIfEmptyT ⟨comma list⟩ {⟨true code⟩}
\clistIfEmptyF ⟨comma list⟩ {⟨false code⟩}
\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.

\clistIfEmptyTF {one,two} {\prgReturn{Empty}} {\prgReturn{NonEmpty}}
\clistIfEmptyTF { , } {\prgReturn{Empty}} {\prgReturn{NonEmpty}} NonEmpty Empty

\clistVarIfEmpty ⟨comma list⟩
\clistVarIfEmptyT ⟨comma list⟩ {⟨true code⟩}
\clistVarIfEmptyF ⟨comma list⟩ {⟨false code⟩}
\clistVarIfEmptyTF ⟨comma list⟩ {⟨true code⟩} {⟨false code⟩}

Tests if the ⟨comma list⟩ is empty (containing no items).

\clistSet \lTmpaClist {one,two}
\clistVarIfEmptyTF \lTmpaClist {\prgReturn{Empty}} {\prgReturn{NonEmpty}}
\clistClear \lTmpaClist
\clistVarIfEmptyTF \lTmpaClist {\prgReturn{Empty}} {\prgReturn{NonEmpty}} NonEmpty Empty
CHAPTER 10. COMMA SEPARATED LISTS (\clist)

\clistIfIn \{⟨comma list⟩\} {⟨item⟩}
\clistIfInT \{⟨comma list⟩\} {⟨item⟩} {⟨true code⟩}
\clistIfInF \{⟨comma list⟩\} {⟨item⟩} {⟨false code⟩}
\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\} {\prgReturn{Yes}} {\prgReturn{No}}
\clistIfInTF \{ a , \{b\} , \{b\}, c \} \{d\} {\prgReturn{Yes}} {\prgReturn{No}}

\clistVarIfIn ⟨comma list⟩ {⟨item⟩}
\clistVarIfInT ⟨comma list⟩ {⟨item⟩} {⟨true code⟩}
\clistVarIfInF ⟨comma list⟩ {⟨item⟩} {⟨false code⟩}
\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.

\clistSet \lTmpaClist {one,two}
\clistVarIfInTF \lTmpaClist {one} {\prgReturn{Yes}} {\prgReturn{No}}
\clistVarIfInTF \lTmpaClist {three} {\prgReturn{Yes}} {\prgReturn{No}}
Chapter 11

Sequences and Stacks (Seq)

11.1 Constant and Scratch Sequences

\cEmptySeq

Constant that is always empty.

\lTmpaSeq \lTmpbSeq \lTmpcSeq \lTmpiSeq \lTmpjSeq \lTmpkSeq

Scratch sequences 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.

\gTmpaSeq \gTmpbSeq \gTmpcSeq \gTmpiSeq \gTmpjSeq \gTmpkSeq

Scratch sequences 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.

11.2 Creating and Using Sequences

\seqNew (sequence)

Creates a new (sequence) or raises an error if the name is already taken. The declaration is global. The (sequence) initially contains no items.

\seqNew \lFooSomeSeq

\seqConstFromClist (seq var) {(comma-list)}

Creates a new constant (seq var) or raises an error if the name is already taken. The (seq var) is set globally to contain the items in the (comma list).

\seqConstFromClist \cFooSomeSeq {one,two,three}

\seqVarJoin (seq var) {(separator)}

Returns the contents of the (seq var), with the (separator) between the items. If the sequence has a single
item, it is returned with no separator, and an empty sequence returns nothing. An error is raised if the variable does not exist or if it is invalid.

```latex
\seqSetSplit \lTmpaSeq {[]} {a|b|c|{de}|f}
\seqVarJoin \lTmpaSeq { and }
```

```
a and b and c and de and f
```

\seqVarJoinExtended \lTmpaSeq { and } {, } {, and }

```
a, b, c, de, and f
```

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

### 11.3 Viewing Sequences

\seqVarLog (sequence)

Writes the entries in the sequence in the log file.

```latex
\seqVarLog \lFooSomeSeq
```

\seqVarShow (sequence)

Displays the entries in the sequence in the terminal.

```latex
\seqVarShow \lFooSomeSeq
```

### 11.4 Setting Sequences

\seqSetFromClist (sequence) (comma-list)

Converts the data in the comma list into a sequence: the original comma list is unchanged.

```latex
\seqSetFromClist \lTmpaSeq {one,two,three}
\seqVarJoin \lTmpaSeq { and }
```

```
one and two and three
```

\seqSetSplit (sequence) {(delimiter)} {(token list)}

Splits the token list into items separated by delimiter, and assigns the result to the sequence. Spaces on both sides of each item are ignored, then one set of outer braces is removed (if any); this space trimming behaviour is identical to that of Clist functions. Empty items are preserved by \seqSetSplit.
and can be removed afterwards using \texttt{\textbackslash seqVarRemoveAll \textbackslash{sequence} \{\}}. The \texttt{(delimiter)} may not contain \{,\} or # (assuming \TeX’s normal category code régime). If the \texttt{(delimiter)} is empty, the \texttt{(token list)} is split into \texttt{(items)} as a \texttt{(token list)}.

\begin{verbatim}
\seqSetSplit \lTmpaSeq {,} {1,2,3} \seqVarJoin \lTmpaSeq { and }
\end{verbatim}

\begin{verbatim}
\seqSetEq \langle \texttt{sequence}1 \rangle \langle \texttt{sequence}2 \rangle \sets\texttt{content of} \langle \texttt{sequence}1 \rangle \texttt{equal to that of} \langle \texttt{sequence}2 \rangle.
\seqSetFromClist \lTmpaSeq \{one,two,three,four\} \seqSetEq \lTmpbSeq \lTmpaSeq \seqVarJoin \lTmpbSeq { and }
\end{verbatim}

\begin{verbatim}
\seqClear \langle \texttt{sequence} \rangle
\seqClear \lTmpaSeq
\seqClearNew \langle \texttt{sequence} \rangle \set\texttt{ensures that the} \langle \texttt{sequence} \rangle \texttt{exists globally by applying} \texttt{\textbackslash seqNew} \texttt{if necessary, then applies} \texttt{\textbackslash seqClear} \texttt{to leave the} \langle \texttt{sequence} \rangle \texttt{empty.}
\seqClearNew \lFooSomeSeq \seqSetFromClist \lFooSomeSeq \{one,two,three\} \seqVarJoin \lFooSomeSeq { and }
\end{verbatim}

\begin{verbatim}
\seqConcat \langle \texttt{sequence}1 \rangle \langle \texttt{sequence}2 \rangle \langle \texttt{sequence}3 \rangle \set\texttt{concatenates the content of} \langle \texttt{sequence}2 \rangle \texttt{and} \langle \texttt{sequence}3 \rangle \texttt{together and saves the result in} \langle \texttt{sequence}1 \rangle. \texttt{The items in} \langle \texttt{sequence}2 \rangle \texttt{are placed at the left side of the new sequence.}
\seqSetFromClist \lTmpbSeq \{one,two\} \seqSetFromClist \lTmpcSeq \{three,four\} \seqConcat \lTmpaSeq \lTmpbSeq \lTmpcSeq \seqVarJoin \lTmpaSeq {, }
\end{verbatim}

\begin{verbatim}
\seqPutLeft \langle \texttt{sequence} \rangle \{\langle \texttt{item} \rangle\} \set\texttt{appends the} \langle \texttt{item} \rangle \texttt{to the left of the} \langle \texttt{sequence} \rangle.
\seqSetFromClist \lTmpaSeq \{one,two\} \seqPutLeft \lTmpaSeq \{zero\} \seqVarJoin \lTmpaSeq { and }
\end{verbatim}

\begin{verbatim}
\seqPutRight \langle \texttt{sequence} \rangle \{\langle \texttt{item} \rangle\} \set\texttt{appends the} \langle \texttt{item} \rangle \texttt{to the right of the} \langle \texttt{sequence} \rangle.
\seqSetFromClist \lTmpaSeq \{one,two\} \seqPutRight \lTmpaSeq \{zero\} \seqVarJoin \lTmpaSeq { and }
\end{verbatim}
11.5 Modifying Sequences

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

\seqVarRemoveDuplicates \langle sequence \rangle

Removes duplicate items from the \langle sequence \rangle, leaving the left most copy of each item in the \langle sequence \rangle. The \langle item \rangle comparison takes place on a token basis, as for \tlIfEqTF.

\seqVarRemoveAll \langle sequence \rangle \{ \langle item \rangle \}

Removes every occurrence of \langle item \rangle from the \langle sequence \rangle. The \langle item \rangle comparison takes place on a token basis, as for \tlIfEqTF.

\seqVarReverse \langle sequence \rangle

Reverses the order of the items stored in the \langle sequence \rangle.

11.6 Working with the Contents of Sequences

\seqVarCount \langle sequence \rangle

Returns the number of items in the \langle sequence \rangle as an \langle integer denotation \rangle. The total number of items in a \langle sequence \rangle includes those which are empty and duplicates, i.e. every item in a \langle sequence \rangle is unique.

\seqVarItem \langle sequence \rangle \{ \langle integer expression \rangle \}

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

\seqSetFromClist \lTmpaSeq {one,two,three,four}
\tlSet \lTmpaTl {\seqVarItem \lTmpaSeq {3}}
\tlUse \lTmpaTl
three

\seqVarRandItem \langle seq var \rangle

Selects a pseudo-random item of the \langle sequence \rangle. If the \langle sequence \rangle is empty the result is empty.

\seqSetFromClist \lTmpaSeq {one,two,three,four,five,six}
\tlSet \lTmpaTl {\seqVarRandItem \lTmpaSeq}
\tlUse \lTmpaTl
\tlSet \lTmpaTl {\seqVarRandItem \lTmpaSeq}
\tlUse \lTmpaTl
three six

11.7 Sequences as Stacks

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

\seqGet \langle sequence \rangle \langle token list variable \rangle

Reads the top item from a \langle sequence \rangle into the \langle token list variable \rangle without removing it from the \langle sequence \rangle. The \langle token list variable \rangle is assigned locally. If \langle sequence \rangle is empty the \langle token list variable \rangle is set to the special marker \qNoValue.

\seqSetFromClist \lTmpaSeq {two,three,four}
\seqGet \lTmpaSeq \lTmpaTl
two
\tlUse \lTmpaTl

\seqGetT \langle sequence \rangle \langle token list variable \rangle {\langle true code \rangle}
\seqGetF \langle sequence \rangle \langle token list variable \rangle {\langle false code \rangle}
\seqGetTF \langle sequence \rangle \langle token list variable \rangle {\langle true code \rangle} {\langle false code \rangle}

If the \langle sequence \rangle is empty, leaves the \langle false code \rangle in the input stream. The value of the \langle token list variable \rangle is not defined in this case and should not be relied upon. If the \langle sequence \rangle is non-empty, stores the top item from a \langle sequence \rangle in the \langle token list variable \rangle without removing it from the \langle sequence \rangle. The \langle token list variable \rangle is assigned locally.

\seqSetFromClist \lTmpaSeq {two,three,four}
\seqGetTF \lTmpaSeq \lTmpaTl {\prgReturn{Yes}} {\prgReturn{No}}
Yes

\seqPop \langle sequence \rangle \langle token list variable \rangle

Pops the top item from a \langle sequence \rangle into the \langle token list variable \rangle. The \langle token list variable \rangle is assigned locally. If \langle sequence \rangle is empty the \langle token list variable \rangle is set to the special marker \qNoValue.
\seqSetFromClist \lTmpaSeq {two,three,four}
\seqPop \lTmpaSeq \lTmpaTl
\seqVarJoin \lTmpaSeq {,}
three,four

\seqPopT \langle sequence \rangle \langle token list variable \rangle \{(true code)\}
\seqPopF \langle sequence \rangle \langle token list variable \rangle \{(false code)\}
\seqPopTF \langle sequence \rangle \langle token list variable \rangle \{(true code)\} \{(false code)\}

If the \langle sequence \rangle is empty, leaves the \langle false code \rangle in the input stream. The value of the \langle token list variable \rangle is not defined in this case and should not be relied upon. If the \langle sequence \rangle is non-empty, pops the top item from the \langle sequence \rangle in the \langle token list variable \rangle, i.e., removes the item from the \langle sequence \rangle. The \langle token list variable \rangle is assigned locally.

\seqPopTF \cEmptySeq \lTmpaTl \{\prgReturn{Yes}\} \{\prgReturn{No}\} No

\seqPush \langle sequence \rangle \{(item)\}

Adds the \{(item)\} to the top of the \langle sequence \rangle.

\seqSetFromClist \lTmpaSeq {two,three,four}
\seqPush \lTmpaSeq \{one\}
\seqVarJoin \lTmpaSeq {,}
one|two|three|four

You can only push one item to the \langle sequence \rangle with \seqPush, which is different from \ClistPush.

11.8 Recovering Items from Sequences

Items can be recovered from either the left or the right of sequences. For implementation reasons, the actions at the left of the sequence are faster than those acting on the right. These functions all assign the recovered material locally.

\seqGetLeft \langle sequence \rangle \langle token list variable \rangle

Stores the left-most item from a \langle sequence \rangle in the \langle token list variable \rangle without removing it from the \langle sequence \rangle. The \langle token list variable \rangle is assigned locally. If \langle sequence \rangle is empty the \langle token list variable \rangle is set to the special marker \qNoValue.

\seqSetFromClist \lTmpaSeq {two,three,four}
\seqGetLeft \lTmpaSeq \lTmpaTl
\tlUse \lTmpaTl
two

\seqGetLeftT \langle sequence \rangle \langle token list variable \rangle \{(true code)\}
\seqGetLeftF \langle sequence \rangle \langle token list variable \rangle \{(false code)\}
\seqGetLeftTF \langle sequence \rangle \langle token list variable \rangle \{(true code)\} \{(false code)\}

If the \langle sequence \rangle is empty, leaves the \langle false code \rangle in the input stream. The value of the \langle token list variable \rangle is not defined in this case and should not be relied upon. If the \langle sequence \rangle is non-empty, stores the left-most item from the \langle sequence \rangle in the \langle token list variable \rangle without removing it from the \langle sequence \rangle, then leaves the \langle true code \rangle in the input stream. The \langle token list variable \rangle is assigned locally.

\seqSetFromClist \lTmpaSeq {two,three,four}
\seqGetLeftTF \lTmpaSeq \lTmpaTl \{\prgReturn{Yes}\} \{\prgReturn{No}\} Yes
\seqGetRight (sequence) (token list variable)

Stores the right-most item from a (sequence) in the (token list variable) without removing it from the (sequence). The (token list variable) is assigned locally. If (sequence) is empty the (token list variable) is set to the special marker \qNoValue.

\seqSetFromClist \lTmpaSeq \{two,three,four\}
\seqGetRight \lTmpaSeq \lTmpaTl
\tlUse \lTmpaTl
four

\seqGetRightT (sequence) (token list variable) \{\langle true code\rangle\}
\seqGetRightF (sequence) (token list variable) \{\langle false code\rangle\}
\seqGetRightTF (sequence) (token list variable) \{\langle true code\rangle\} \{\langle false code\rangle\}

If the (sequence) is empty, leaves the (false code) in the input stream. The value of the (token list variable) is not defined in this case and should not be relied upon. If the (sequence) is non-empty, stores the right-most item from the (sequence) in the (token list variable) without removing it from the (sequence), then leaves the (true code) in the input stream. The (token list variable) is assigned locally.

\seqSetFromClist \lTmpaSeq \{two,three,four\}
\seqGetRightTF \lTmpaSeq \lTmpaTl \{\prgReturn{Yes}\} \{\prgReturn{No}\}

\seqPopLeft (sequence) (token list variable)

Pops the left-most item from a (sequence) into the (token list variable), i.e., removes the item from the sequence and stores it in the (token list variable). The assignment of the (token list variable) is local. If (sequence) is empty the (token list variable) is set to the special marker \qNoValue.

\seqSetFromClist \lTmpaSeq \{two,three,four\}
\seqPopLeft \lTmpaSeq \lTmpaTl \seqVarJoin \lTmpaSeq \{,\}
three,four

\seqPopLeftT (sequence) (token list variable) \{\langle true code\rangle\}
\seqPopLeftF (sequence) (token list variable) \{\langle false code\rangle\}
\seqPopLeftTF (sequence) (token list variable) \{\langle true code\rangle\} \{\langle false code\rangle\}

If the (sequence) is empty, leaves the (false code) in the input stream. The value of the (token list variable) is not defined in this case and should not be relied upon. If the (sequence) is non-empty, pops the left-most item from the (sequence) in the (token list variable), i.e., removes the item from the (sequence), then leaves the (true code) in the input stream. The (token list variable) is assigned locally.

\seqSetFromClist \lTmpaSeq \{two,three,four\}
\seqPopLeftTF \cEmptySeq \lTmpaTl \{\prgReturn{Yes}\} \{\prgReturn{No}\}
No

\seqPopRight (sequence) (token list variable)

Pops the right-most item from a (sequence) into the (token list variable), i.e., removes the item from the sequence and stores it in the (token list variable). The assignment of the (token list variable) is local. If (sequence) is empty the (token list variable) is set to the special marker \qNoValue.

\seqSetFromClist \lTmpaSeq \{two,three,four\}
\seqPopRight \lTmpaSeq \lTmpaTl \seqVarJoin \lTmpaSeq \{,\}
two,three
CHAPTER 11. SEQUENCES AND STACKS (SEQ)

\seqPopRightT (sequence) (token list variable) {⟨true code⟩}
\seqPopRightF (sequence) (token list variable) {⟨false code⟩}
\seqPopRightTF (sequence) (token list variable) {⟨true code⟩} {⟨false code⟩}

If the ⟨sequence⟩ is empty, leaves the ⟨false code⟩ in the input stream. The value of the ⟨token list variable⟩ is not defined in this case and should not be relied upon. If the ⟨sequence⟩ is non-empty, pops the right-most item from the ⟨sequence⟩ in the ⟨token list variable⟩, i.e., removes the item from the ⟨sequence⟩, then leaves the ⟨true code⟩ in the input stream. The ⟨token list variable⟩ is assigned locally.

\seqPopRightTF \cEmptySeq \lTmpaTl {\prgReturn{Yes}} {\prgReturn{No}}  

No

11.9 Mapping over Sequences

\seqVarMapInline (sequence) {⟨inline function⟩}

Applies ⟨inline function⟩ to every ⟨item⟩ stored within the ⟨sequence⟩. The ⟨inline function⟩ should consist of code which will receive the ⟨item⟩ as #1. The ⟨items⟩ are returned from left to right.

\IgnoreSpacesOn
\seqSetFromClist \lTmpkSeq {one,two,three}
\tlClear \lTmpaTl
\seqVarMapInline \lTmpkSeq {
    \tlPutRight \lTmpaTl {(#1)}
}
\tlUse \lTmpaTl
\IgnoreSpacesOff

(one)(two)(three)

\seqVarMapVariable (sequence) (variable) {⟨code⟩}

Stores each ⟨item⟩ of the ⟨sequence⟩ 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 ⟨sequence⟩, or its original value if the ⟨sequence⟩ is empty. The ⟨items⟩ are returned from left to right.

\IgnoreSpacesOn
\intZero \lTmpaInt
\seqSetFromClist \lTmpaSeq {1,3,7}
\seqVarMapVariable \lTmpaSeq \lTmpiTl {
    \intAdd \lTmpaInt {\lTmpiTl*\lTmpiTl}
}
\intUse \lTmpaInt
\IgnoreSpacesOff

59

11.10 Sequence Conditionals

\seqIfExist ⟨sequence⟩
\seqIfExistT (sequence) {⟨true code⟩}
\seqIfExistF (sequence) {⟨false code⟩}
\seqIfExistTF (sequence) {⟨true code⟩} {⟨false code⟩}

Tests whether the ⟨sequence⟩ is currently defined. This does not check that the ⟨sequence⟩ really is a sequence variable.
CHAPTER 11. SEQUENCES AND STACKS (SEQ)

\seqIfExistTF \lTmpaSeq {\prgReturn{Yes}} {\prgReturn{No}}
\seqIfExistTF \lFooUndefinedSeq {\prgReturn{Yes}} {\prgReturn{No}}

Yes No

\seqVarIfEmpty ⟨sequence⟩
\seqVarIfEmptyT ⟨sequence⟩ {⟨true code⟩}
\seqVarIfEmptyF ⟨sequence⟩ {⟨false code⟩}
\seqVarIfEmptyTF ⟨sequence⟩ {⟨true code⟩} {⟨false code⟩}

Tests if the ⟨sequence⟩ is empty (containing no items).

\seqSetFromClist \lTmpaSeq {one,two}
\seqVarIfEmptyTF \lTmpaSeq {\prgReturn{Empty}} {\prgReturn{NonEmpty}}
\seqClear \lTmpaSeq
\seqVarIfEmptyTF \lTmpaSeq {\prgReturn{Empty}} {\prgReturn{NonEmpty}}

NonEmpty Empty

\seqVarIfIn ⟨sequence⟩ {⟨item⟩}
\seqVarIfInT ⟨sequence⟩ {⟨item⟩} {⟨true code⟩}
\seqVarIfInF ⟨sequence⟩ {⟨item⟩} {⟨false code⟩}
\seqVarIfInTF ⟨sequence⟩ {⟨item⟩} {⟨true code⟩} {⟨false code⟩}

Tests if the ⟨item⟩ is present in the ⟨sequence⟩.

\seqSetFromClist \lTmpaSeq {one,two}
\seqVarIfInTF \lTmpaSeq {one} {\prgReturn{Yes}} {\prgReturn{Not}}
\seqVarIfInTF \lTmpaSeq {three} {\prgReturn{Yes}} {\prgReturn{Not}}
Chapter 12

Property Lists (Prop)

\[\text{\LaTeX}3\] implements a “property list” data type, which contain an unordered list of entries each of which consists of a \langle key\rangle and an associated \langle value\rangle. The \langle key\rangle and \langle value\rangle may both be any \langle balanced text\rangle, the \langle key\rangle is processed using \texttt{\textbackslash t1ToStr}, meaning that category codes are ignored. It is possible to map functions to property lists such that the function is applied to every key–value pair within the list.

Each entry in a property list must have a unique \langle key\rangle: if an entry is added to a property list which already contains the \langle key\rangle then the new entry overwrites the existing one. The \langle keys\rangle are compared on a string basis, using the same method as \texttt{\textbackslash strIfEq}.

12.1 Constant and Scratch Sequences

\texttt{\textbackslash cEmptyProp}\\
\text{Constant that is always empty.}\\
\texttt{\textbackslash lTmpaProp \textbackslash lTmpbProp \textbackslash lTmpcProp \textbackslash lTmpiProp \textbackslash lTmpjProp \textbackslash lTmpkProp}\\
\text{Scratch property 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.}\\
\texttt{\textbackslash gTmpaProp \textbackslash gTmpbProp \textbackslash gTmpcProp \textbackslash gTmpiProp \textbackslash gTmpjProp \textbackslash gTmpkProp}\\
\text{Scratch property 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.}

12.2 Creating and Using Property Lists

\texttt{\textbackslash propNew (property list)}\\
\text{Creates a new \langle property list\rangle or raises an error if the name is already taken. The declaration is global. The \langle property list\rangle initially contains no entries.}\\
\texttt{\textbackslash propNew \textbackslash lFooSomeProp}
CHAPTER 12. PROPERTY LISTS (PROP) 82

\propConstFromKeyval \(prop\ var\)  
\{
  \langle key1 \rangle = \langle value1 \rangle ,
  \langle key2 \rangle = \langle value2 \rangle , \ldots
\}  

Creates a new constant \(prop\ var\) or raises an error if the name is already taken. The \(prop\ var\) is set globally to contain key–value pairs given in the second argument, processed in the way described for \propSetFromKeyval. If duplicate keys appear only the last of the values is kept. This function correctly detects the = and , signs provided they have the standard category code 12 or they are active.

\propConstFromKeyval \cFooSomeProp \{key1=one,key2=two,key3=three\}

\propToKeyval \(property\ list\)  

Returns the \(property\ list\) in a key–value notation. Keep in mind that a \(property\ list\) is unordered, while key–value interfaces don’t necessarily are, so this can’t be used for arbitrary interfaces.

\propToKeyval \lTmpaProp

12.3 Viewing Property Lists

\propVarLog \(property\ list\)  

Writes the entries in the \(property\ list\) in the log file.

\propVarLog \lTmpaProp

\propVarShow \(property\ list\)  

Displays the entries in the \(property\ list\) in the terminal.

\propVarShow \lTmpaProp

12.4 Setting Property Lists

\propSetFromKeyval \(prop\ var\)  
\{
  \langle key1 \rangle = \langle value1 \rangle ,
  \langle key2 \rangle = \langle value2 \rangle , \ldots
\}  

Sets \(prop\ var\) to contain key–value pairs given in the second argument. If duplicate keys appear only the last of the values is kept. Spaces are trimmed around every \langle key \rangle and every \langle value \rangle, and if the result of trimming spaces consists of a single brace group then a set of outer braces is removed. This enables both the \langle key \rangle and the \langle value \rangle to contain spaces, commas or equal signs. The \langle key \rangle is then processed by \tlToStr. This function correctly detects the = and , signs provided they have the standard category code 12 or they are active.

\propSetFromKeyval \lTmpaProp \{key1=one,key2=two\}
\propSetEq \langle \text{property list}_1 \rangle \langle \text{property list}_2 \rangle

Sets the content of \langle \text{property list}_1 \rangle equal to that of \langle \text{property list}_2 \rangle.

\propSetFromKeyval \lTmpaProp \{key1=one,key2=two,key3=three\}
\propSetEq \lTmpbProp \lTmpaProp
\propVarLog \lTmpbProp

\propClear \langle \text{property list} \rangle

Clears all entries from the \langle property list \rangle.

\propClear \lTmpaProp

\propClearNew \langle \text{property list} \rangle

Ensures that the \langle property list \rangle exists globally by applying \propNew if necessary, then applies \propClear to leave the list empty.

\propClearNew \lFooSomeProp

\propConcat \langle \text{prop var}_1 \rangle \langle \text{prop var}_2 \rangle \langle \text{prop var}_3 \rangle

Combines the key–value pairs of \langle \text{prop var}_2 \rangle and \langle \text{prop var}_3 \rangle, and saves the result in \langle \text{prop var}_1 \rangle. If a key appears in both \langle \text{prop var}_2 \rangle and \langle \text{prop var}_3 \rangle then the last value, namely the value in \langle \text{prop var}_3 \rangle is kept.

\propSetFromKeyval \lTmpaProp \{key1=one,key2=two\}
\propSetFromKeyval \lTmpbProp \{key3=three,key4=four\}
\propConcat \lTmpaProp \lTmpbProp \lTmpcProp
\propVarLog \lTmpaProp

\propPut \langle \text{property list} \rangle \{(\langle \text{key} \rangle \} \{(\langle \text{value} \rangle \}

Adds an entry to the \langle property list \rangle which may be accessed using the \langle key \rangle and which has \langle value \rangle. If the \langle key \rangle is already present in the \langle property list \rangle, the existing entry is overwritten by the new \langle value \rangle. Both the \langle key \rangle and \langle value \rangle may contain any \langle balanced text \rangle. The \langle key \rangle is stored after processing with \tlToStr, meaning that category codes are ignored.

\propSetFromKeyval \lTmpaProp \{key1=one,key2=two\}
\propPut \lTmpaProp \{key1\} \{newone\}
\propVarLog \lTmpaProp

\propPutIfNew \langle \text{property list} \rangle \{(\langle \text{key} \rangle \} \{(\langle \text{value} \rangle \}

If the \langle key \rangle is present in the \langle property list \rangle then no action is taken. Otherwise, a new entry is added as described for \propPut.

\propSetFromKeyval \lTmpaProp \{key1=one,key2=two\}
\propPutIfNew \lTmpaProp \{key1\} \{newone\}
\propVarLog \lTmpaProp
CHAPTER 12. PROPERTY LISTS (PROP)

\propPutFromKeyval \(\langle \text{prop var} \rangle\)
{
  \(\langle \text{key1} \rangle = \langle \text{value1} \rangle,\)
  \(\langle \text{key2} \rangle = \langle \text{value2} \rangle, \ldots\)
}

Updates the \(\langle \text{prop var} \rangle\) by adding entries for each key–value pair given in the second argument. The addition is done through \propPut, hence if the \(\langle \text{prop var} \rangle\) already contains some of the keys, the corresponding values are discarded and replaced by those given in the key–value list. If duplicate keys appear in the key–value list then only the last of the values is kept.

\propSetFromKeyval \lTmpaProp \{key1=one,key2=two\}
\propPutFromKeyval \lTmpaProp \{key1=newone,key3=three\}
\propVarLog \lTmpaProp

\propVarRemove \(\langle \text{property list} \rangle\) \{(\text{key})\}

Removes the entry listed under \(\langle \text{key} \rangle\) from the \(\langle \text{property list} \rangle\). If the \(\langle \text{key} \rangle\) is not found in the \(\langle \text{property list} \rangle\) no change occurs, i.e., there is no need to test for the existence of a key before deleting it.

\propSetFromKeyval \lTmpaProp \{key1=one,key2=two,key3=three\}
\propVarRemove \lTmpaProp \{key2\}
\propVarLog \lTmpaProp

12.5 Recovering Values from Property Lists

\propVarCount \(\langle \text{property list} \rangle\)

Returns the number of key–value pairs in the \(\langle \text{property list} \rangle\) as an \(\langle \text{integer denotation} \rangle\).

\propSetFromKeyval \lTmpaProp \{key1=one,key2=two,key3=three\}
\propVarCount \lTmpaProp

\propVarItem \(\langle \text{property list} \rangle\) \{(\text{key})\}

Returns the \(\langle \text{value} \rangle\) corresponding to the \(\langle \text{key} \rangle\) in the \(\langle \text{property list} \rangle\). If the \(\langle \text{key} \rangle\) is missing, nothing is returned.

\propSetFromKeyval \lTmpaProp \{key1=one,key2=two,key3=three\}
\tlSet \lTmpaTl \{\propVarItem \lTmpaProp \{key2\}\}
\tlUse \lTmpaTl
two

\propGet \(\langle \text{property list} \rangle\) \{(\text{key})\} \(\langle \text{token list variable} \rangle\)

Recovers the \(\langle \text{value} \rangle\) stored with \(\langle \text{key} \rangle\) from the \(\langle \text{property list} \rangle\), and places this in the \(\langle \text{token list variable} \rangle\). If the \(\langle \text{key} \rangle\) is not found in the \(\langle \text{property list} \rangle\) then the \(\langle \text{token list variable} \rangle\) is set to the special marker \qNoValue. The assignment of the \(\langle \text{token list variable} \rangle\) is local.

\propSetFromKeyval \lTmpaProp \{key1=one,key2=two,key3=three\}
\propGet \lTmpaProp \{key2\} \lTmpaTl
two
\tlUse \lTmpaTl
CHAPTER 12. PROPERTY LISTS (PROP)

\propGetT \langle property list \rangle \{ \langle key \rangle \} \langle token list variable \rangle \{ \langle true code \rangle \}
\propGetF \langle property list \rangle \{ \langle key \rangle \} \langle token list variable \rangle \{ \langle false code \rangle \}
\propGetTF \langle property list \rangle \{ \langle key \rangle \} \langle token list variable \rangle \{ \langle true code \rangle \} \{ \langle false code \rangle \}

If the \langle key \rangle is not present in the \langle property list \rangle, leaves the \langle false code \rangle in the input stream. The value of the \langle token list variable \rangle is not defined in this case and should not be relied upon. If the \langle key \rangle is present in the \langle property list \rangle, stores the corresponding \langle value \rangle in the \langle token list variable \rangle without removing it from the \langle property list \rangle, then leaves the \langle true code \rangle in the input stream. The \langle token list variable \rangle is assigned locally.

\propSetFromKeyval \lTmpaProp \{ key1=one, key2=two, key3=three \}
\propGetTF \lTmpaProp \{ key2 \} \lTmpaTl \{ \prgReturn{Yes} \} \{ \prgReturn{No} \}
Yes

\propPop \langle property list \rangle \{ \langle key \rangle \} \langle token list variable \rangle
Recoveres the \langle value \rangle stored with \langle key \rangle from the \langle property list \rangle, and places this in the \langle token list variable \rangle. If the \langle key \rangle is not found in the \langle property list \rangle then the \langle token list variable \rangle is set to the special marker \qNoValue. The \langle key \rangle and \langle value \rangle are then deleted from the property list. The assignment of the \langle token list variable \rangle is local.

\propSetFromKeyval \lTmpaProp \{ key1=one, key2=two, key3=three \}
\propPop \lTmpaProp \{ key2 \} \lTmpaTl
Pop: \tlUse \lTmpaTl.
Count: \propVarCount \lTmpaProp.

\propPopT \langle property list \rangle \{ \langle key \rangle \} \langle token list variable \rangle \{ \langle true code \rangle \}
\propPopF \langle property list \rangle \{ \langle key \rangle \} \langle token list variable \rangle \{ \langle false code \rangle \}
\propPopTF \langle property list \rangle \{ \langle key \rangle \} \langle token list variable \rangle \{ \langle true code \rangle \} \{ \langle false code \rangle \}

If the \langle key \rangle is not present in the \langle property list \rangle, leaves the \langle false code \rangle in the input stream. The value of the \langle token list variable \rangle is not defined in this case and should not be relied upon. If the \langle key \rangle is present in the \langle property list \rangle, pops the corresponding \langle value \rangle in the \langle token list variable \rangle, i.e. removes the item from the \langle token list variable \rangle is assigned locally.

\propSetFromKeyval \lTmpaProp \{ key1=one, key2=two, key3=three \}
\propPopTF \lTmpaProp \{ key2 \} \lTmpaTl \{ \prgReturn{Yes} \} \{ \prgReturn{No} \}
Yes

12.6 Mapping over property lists

\propVarMapInline \langle property list \rangle \{ \langle inline function \rangle \}
Applies \langle inline function \rangle to every \langle entry \rangle stored within the \langle property list \rangle. The \langle inline function \rangle should consist of code which receives the \langle key \rangle as \#1 and the \langle value \rangle as \#2. The order in which \langle entries \rangle are returned is not defined and should not be relied upon.
12.7 Property List Conditionals

\propIfExist ⟨property list⟩
\propIfExistT ⟨property list⟩ {⟨true code⟩}
\propIfExistF ⟨property list⟩ {⟨false code⟩}
\propIfExistTF ⟨property list⟩ {⟨true code⟩} {⟨false code⟩}

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

\propIfExistTF \lTmpaProp {\prgReturn{Yes}} {\prgReturn{No}}
\propIfExistTF \lFooUndefinedProp {\prgReturn{Yes}} {\prgReturn{No}}

\propVarIfEmpty ⟨property list⟩
\propVarIfEmptyT ⟨property list⟩ {⟨true code⟩}
\propVarIfEmptyF ⟨property list⟩ {⟨false code⟩}
\propVarIfEmptyTF ⟨property list⟩ {⟨true code⟩} {⟨false code⟩}

Tests if the ⟨property list⟩ is empty (containing no entries).

\propVarIfEmptyT \lTmpaProp {key1=one,key2=two} {\prgReturn{Empty}} {\prgReturn{NonEmpty}}
\propClear \lTmpaProp
\propVarIfEmptyTF \lTmpaProp {\prgReturn{Empty}} {\prgReturn{NonEmpty}}
NonEmpty Empty

\propVarIfIn ⟨property list⟩ {⟨key⟩}
\propVarIfInT ⟨property list⟩ {⟨key⟩} {⟨true code⟩}
\propVarIfInF ⟨property list⟩ {⟨key⟩} {⟨false code⟩}
\propVarIfInTF ⟨property list⟩ {⟨key⟩} {⟨true code⟩} {⟨false code⟩}

Tests if the ⟨key⟩ is present in the ⟨property list⟩, making the comparison using the method described by \strIfEqTF.

\propVarIfInTF \lTmpaProp {key1} {\prgReturn{Yes}} {\prgReturn{Not}}
\propVarIfInTF \lTmpaProp {key3} {\prgReturn{Yes}} {\prgReturn{Not}}
Yes Not
Chapter 13

Regular Expressions (Regex)

This module provides regular expression testing, extraction of submatches, splitting, and replacement, all acting on token lists. The syntax of regular expressions is mostly a subset of the PCRE syntax (and very close to POSIX), with some additions due to the fact that \TeX manipulates tokens rather than characters. For performance reasons, only a limited set of features are implemented. Notably, back-references are not supported.

Let us give a few examples. The following example replace the first occurrence of “at” with “is” in the token list variable \lTmpaTl.

\tlSet \lTmpaTl {That cat.}
\regexReplaceOnce {at} {is} \lTmpaTl
\tlUse \lTmpaTl

This cat.

A more complicated example is a pattern to emphasize each word and add a comma after it:

\tlSet \lTmpaTl {That cat.}
\regexReplaceAll {\w+} {\c{underline} \cB\{ \0 \cE\} ,} \lTmpaTl
\tlUse \lTmpaTl

That, cat.

The \w sequence represents any “word” character, and + indicates that the \w sequence should be repeated as many times as possible (at least once), hence matching a word in the input token list. In the replacement text, \0 denotes the full match (here, a word). The command \underline is inserted using \c{underline}, and its argument \0 is put between braces \cB\{ and \cE\}.

If a regular expression is to be used several times, it can be compiled once, and stored in a regex variable using \regexSet. For example,

\regexNew \lFooReg
\regexSet \lFooReg {\c{begin} \cB\{ \c{[\^BE].*} \cE\} .} \lTmpaTl

stores in \lFooReg a regular expression which matches the starting marker for an environment: \begin, followed by a begin-group token (\cB.), then any number of tokens which are neither begin-group nor end-group character tokens (\c{[\^BE].*}), ending with an end-group token (\cE.). As explained later, the parentheses “capture” the result of \c{[\^BE].*}, giving us access to the name of the environment when doing replacements.

13.1 Regular Expression Variables

If a regular expression is to be used several times, it is better to compile it once rather than doing it each time the regular expression is used. The compiled regular expression is stored in a variable. All of this module’s functions can be given their regular expression argument either as an explicit string or as a compiled regular expression.
Scratch regex variables for local assignment. These are never used by function package, and so are safe for use with any function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

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

Creates a new (regex var) or raises an error if the name is already taken. The declaration is global. The (regex var) is initially such that it never matches.

Stores a compiled version of the (regular expression) in the (regex var). For instance, this function can be used as

```
\regexNew \lMyRegex
\regexSet \lMyRegex {my\ (simple\ )? reg(ular\ expression)}
```

Creates a new constant (regex var) or raises an error if the name is already taken. The value of the (regex var) is set globally to the compiled version of the (regular expression).

```
\regexLog
\regexVarLog (regex var)
\regexShow
\regexVarShow (regex var)
```

Displays in the terminal or writes in the log file (respectively) how l3regex interprets the (regex). For instance, \regexShow \{\A X|Y\} shows

```
+-branch
  anchor at start (\A)
  char code 88 (X)
+-branch
  char code 89 (Y)
```

indicating that the anchor \A only applies to the first branch: the second branch is notanchored to the beginning of the match.

### 13.2 Regular Expression Matching

```
\regexMatch \{\langle regex\rangle\} \{\langle token\ list\rangle\}
\regexMatchT \{\langle regex\rangle\} \{\langle token\ list\rangle\} \{\langle true\ code\rangle\}
\regexMatchF \{\langle regex\rangle\} \{\langle token\ list\rangle\} \{\langle false\ code\rangle\}
\regexMatchTF \{\langle regex\rangle\} \{\langle token\ list\rangle\} \{\langle true\ code\rangle\} \{\langle false\ code\rangle\}
```

Tests whether the (regular expression) matches any part of the (token list). For instance,
CHAPTER 13. REGULAR EXPRESSIONS (REGEX)

```
\regexMatchTF {b [cde]*} {abecdcx} {\prgPrint{True}} {\prgPrint{False}}
\regexMatchTF {[b-dq-w]} {example} {\prgPrint{True}} {\prgPrint{False}}
```

True False

Tests whether the ⟨regex var⟩ matches any part of the ⟨token list⟩.

```
\regexVarMatch ⟨regex var⟩ {⟨token list⟩}
\regexVarMatchTF ⟨regex var⟩ {⟨true code⟩}
\regexVarMatchFP ⟨regex var⟩ {⟨false code⟩}
\regexVarMatchTF ⟨regex var⟩ {⟨token list⟩} {⟨true code⟩} {⟨false code⟩}
```

Sets (int var) within the current T\TeX group level equal to the number of times ⟨regular expression⟩ appears in ⟨token list⟩. The search starts by finding the left-most longest match, respecting greedy and lazy (non-greedy) operators. Then the search starts again from the character following the last character of the previous match, until reaching the end of the token list. Infinite loops are prevented in the case where the regular expression can match an empty token list: then we count one match between each pair of characters. For instance,

```
\intNew \lFooInt
\regexCount {(b+|c)} {abbababcb} \lFooInt
\intUse \lFooInt
```

5

Determines which of the ⟨regular expressions⟩ matches at the earliest point in the ⟨token list⟩, and leaves the corresponding ⟨code⟩. If several ⟨regex⟩ match starting at the same point, then the first one in the list is selected and the others are discarded. Each ⟨regex⟩ can either be given as a regex variable or as an explicit regular expression.

In detail, for each starting position in the ⟨token list⟩, each of the ⟨regex⟩ is searched in turn. If one of them matches then the corresponding ⟨code⟩ is used and everything else is discarded, while if none of the ⟨regex⟩ match at a given position then the next starting position is attempted. If none of the ⟨regex⟩ match anywhere in the ⟨token list⟩ then nothing is left in the input stream. Note that this differs from nested \regexMatch statements since all ⟨regex⟩ are attempted at each position rather than attempting to match ⟨regex1⟩ at every position before moving on to ⟨regex2⟩.

```
\regexMatchCase
{  
  ⟨⟨regex⟩1⟩ {⟨code case1⟩}
  ⟨⟨regex⟩2⟩ {⟨code case2⟩}
  ...
  ⟨⟨regex⟩n⟩ {⟨code case_n⟩}
} {⟨token list⟩}
```

Determines which of the ⟨regular expressions⟩ matches at the earliest point in the ⟨token list⟩, and leaves the corresponding ⟨code⟩ followed by the ⟨true code⟩ in the input stream. If several ⟨regex⟩ match
starting at the same point, then the first one in the list is selected and the others are discarded. Each ⟨regex⟩ can either be given as a regex variable or as an explicit regular expression.

\regexMatchCaseF  
\{  
  \{ ⟨regex1⟩ \} \{ ⟨code case1⟩ \}  
  \{ ⟨regex2⟩ \} \{ ⟨code case2⟩ \}  
  \ldots  
  \{ ⟨regexn⟩ \} \{ ⟨code case_n⟩ \}  
\} \{ ⟨token list⟩ \}  
\{ ⟨false code⟩ \}

Determines which of the ⟨regular expressions⟩ matches at the earliest point in the ⟨token list⟩, and leaves the corresponding ⟨code⟩. If several ⟨regex⟩ match starting at the same point, then the first one in the list is selected and the others are discarded. If none of the ⟨regex⟩ match, the ⟨false code⟩ is left in the input stream. Each ⟨regex⟩ can either be given as a regex variable or as an explicit regular expression.

\regexMatchCaseTF  
\{  
  \{ ⟨regex1⟩ \} \{ ⟨code case1⟩ \}  
  \{ ⟨regex2⟩ \} \{ ⟨code case2⟩ \}  
  \ldots  
  \{ ⟨regexn⟩ \} \{ ⟨code case_n⟩ \}  
\} \{ ⟨token list⟩ \}  
\{ ⟨true code⟩ \}  
\{ ⟨false code⟩ \}

Determines which of the ⟨regular expressions⟩ matches at the earliest point in the ⟨token list⟩, and leaves the corresponding ⟨code⟩ followed by the ⟨true code⟩ in the input stream. If several ⟨regex⟩ match starting at the same point, then the first one in the list is selected and the others are discarded. If none of the ⟨regex⟩ match, the ⟨false code⟩ is left in the input stream. Each ⟨regex⟩ can either be given as a regex variable or as an explicit regular expression.

### 13.3 Regular Expression Submatch Extraction

\regexExtractOnce  \{ ⟨regex⟩ \} \{ ⟨token list⟩ \} \{ ⟨seq var⟩ \}  
\regexExtractOnceT  \{ ⟨regex⟩ \} \{ ⟨token list⟩ \} \{ ⟨seq var⟩ \} \{ ⟨true code⟩ \}  
\regexExtractOnceF  \{ ⟨regex⟩ \} \{ ⟨token list⟩ \} \{ ⟨seq var⟩ \} \{ ⟨false code⟩ \}  
\regexExtractOnceTF  \{ ⟨regex⟩ \} \{ ⟨token list⟩ \} \{ ⟨seq var⟩ \} \{ ⟨true code⟩ \} \{ ⟨false code⟩ \}

Finds the first match of the ⟨regular expression⟩ in the ⟨token list⟩. If it exists, the match is stored as the first item of the ⟨seq var⟩, and further items are the contents of capturing groups, in the order of their opening parenthesis. The ⟨seq var⟩ is assigned locally. If there is no match, the ⟨seq var⟩ is cleared. The testing versions insert the ⟨true code⟩ into the input stream if a match was found, and the ⟨false code⟩ otherwise.

For instance, assume that you type

\regexExtractOnce \{ \A(La)?TeX(!*)\Z \} \{LaTeX!!!\} \lTmpaSeq

Then the regular expression (anchored at the start with \A and at the end with \Z) must match the whole token list. The first capturing group, (La)?, matches La, and the second capturing group, (!*), matches !!! Thus, \lTmpaSeq contains as a result the items \{LaTeX!!!\}, \{La\}, and \{!!!\}. Note that the n-th item of \lTmpaSeq, as obtained using \seqVarItem, correspond to the submatch numbered (n − 1) in functions such as \regexReplaceOnce.
\regexVarExtractOnce \regexVar \tokenList \seqVar
\regexVarExtractOnceT \regexVar \tokenList \seqVar \trueCode
\regexVarExtractOnceF \regexVar \tokenList \seqVar \falseCode
\regexVarExtractOnceTF \regexVar \tokenList \seqVar \falseCode
\regexVarExtractOnceF \regexVar \tokenList \seqVar \falseCode
\regexVarExtractOnceT \regexVar \tokenList \seqVar \falseCode
\regexVarExtractOnce \regexVar \tokenList \seqVar
\regexVarExtractOnceT \regexVar \tokenList \seqVar \trueCode
\regexVarExtractOnceF \regexVar \tokenList \seqVar \falseCode
\regexVarExtractOnceTF \regexVar \tokenList \seqVar \falseCode

Finds the first match of the \regexVar in the \tokenList. If it exists, the match is stored as the first item of the \seqVar, and further items are the contents of capturing groups, in the order of their opening parenthesis. The \seqVar is assigned locally. If there is no match, the \seqVar is cleared. The testing versions insert the \trueCode into the input stream if a match was found, and the \falseCode otherwise.

\regexExtractAll \regex \tokenList \seqVar
\regexExtractAllT \regex \tokenList \seqVar \trueCode
\regexExtractAllF \regex \tokenList \seqVar \falseCode
\regexExtractAllTF \regex \tokenList \seqVar \falseCode
\regexExtractAll \regex \tokenList \seqVar
\regexExtractAllT \regex \tokenList \seqVar \trueCode
\regexExtractAllF \regex \tokenList \seqVar \falseCode
\regexExtractAllTF \regex \tokenList \seqVar \falseCode

Finds all matches of the \regex in the \tokenList, and stores all the submatch information in a single sequence (concatenating the results of multiple \regexExtractOnce calls). The \seqVar is assigned locally. If there is no match, the \seqVar is cleared. The testing versions insert the \trueCode into the input stream if a match was found, and the \falseCode otherwise. For instance, assume that you type

\regexExtractAll \w+ Hello, world! \lmTmpaSeq

Then the regular expression matches twice, the resulting sequence contains the two items \{Hello\} and \{world\}.

\regexVarExtractAll \regex \tokenList \seqVar
\regexVarExtractAllT \regex \tokenList \seqVar \trueCode
\regexVarExtractAllF \regex \tokenList \seqVar \falseCode
\regexVarExtractAllTF \regex \tokenList \seqVar \falseCode
\regexVarExtractAll \regex \tokenList \seqVar
\regexVarExtractAllT \regex \tokenList \seqVar \trueCode
\regexVarExtractAllF \regex \tokenList \seqVar \falseCode
\regexVarExtractAllTF \regex \tokenList \seqVar \falseCode

Finds all matches of the \regex in the \tokenList, and stores all the submatch information in a single sequence (concatenating the results of multiple \regexVarExtractOnce calls). The \seqVar is assigned locally. If there is no match, the \seqVar is cleared. The testing versions insert the \trueCode into the input stream if a match was found, and the \falseCode otherwise.

\regexSplit \regularExpression \tokenList \seqVar
\regexSplitT \regularExpression \tokenList \seqVar \trueCode
\regexSplitF \regularExpression \tokenList \seqVar \falseCode
\regexSplitTF \regularExpression \tokenList \seqVar \falseCode
\regexSplit \regularExpression \tokenList \seqVar
\regexSplitT \regularExpression \tokenList \seqVar \trueCode
\regexSplitF \regularExpression \tokenList \seqVar \falseCode
\regexSplitTF \regularExpression \tokenList \seqVar \falseCode

Splits the \tokenList into a sequence of parts, delimited by matches of the \regularExpression. If the \regularExpression has capturing groups, then the token lists that they match are stored as items of the sequence as well. The assignment to \seqVar is local. If no match is found the resulting \seqVar has the \tokenList as its sole item. If the \regularExpression matches the empty token list, then the \tokenList is split into single tokens. The testing versions insert the \trueCode into the input stream if a match was found, and the \falseCode otherwise. For example, after

\seqNev \lPathSeq
\regexSplit // the/path/for/this/file.tex \lPathSeq

the sequence \lPathSeq contains the items \{the\}, \{path\}, \{for\}, \{this\}, and \{file.tex\}.
Splits the ⟨token list⟩ into a sequence of parts, delimited by matches of the ⟨regular expression⟩. If the ⟨regex var⟩ has capturing groups, then the token lists that they match are stored as items of the sequence as well. The assignment to ⟨seq var⟩ is local. If no match is found the resulting ⟨seq var⟩ has the ⟨token list⟩ as its sole item. If the ⟨regular expression⟩ matches the empty token list, then the ⟨token list⟩ is split into single tokens. The testing versions insert the ⟨true code⟩ into the input stream if a match was found, and the ⟨false code⟩ otherwise.

13.4 Regular Expression Replacement

Searches for the ⟨regular expression⟩ in the contents of the ⟨tl var⟩ and replaces the first match with the ⟨replacement⟩. In the ⟨replacement⟩, \0 represents the full match, \1 represent the contents of the first capturing group, \2 of the second, etc. The result is assigned locally to ⟨tl var⟩.

Searches for the ⟨regex var⟩ in the contents of the ⟨tl var⟩ and replaces the first match with the ⟨replacement⟩. In the ⟨replacement⟩, \0 represents the full match, \1 represent the contents of the first capturing group, \2 of the second, etc. The result is assigned locally to ⟨tl var⟩.

Replaces all occurrences of the ⟨regex var⟩ in the contents of the ⟨tl var⟩ by the ⟨replacement⟩, where \0 represents the full match, \1 represent the contents of the first capturing group, \2 of the second, etc. Every match is treated independently, and matches cannot overlap. The result is assigned locally to ⟨tl var⟩.

Replaces all occurrences of the ⟨regular expression⟩ in the contents of the ⟨tl var⟩ by the ⟨replacement⟩, where \0 represents the full match, \1 represent the contents of the first capturing group, \2 of the second, etc. Every match is treated independently, and matches cannot overlap. The result is assigned locally to ⟨tl var⟩.
\regexReplaceCaseOnce
\{
   \{\regex\} \{\replacement\}
   \{\regex\} \{\replacement\}
   \ldots
   \{\regex\} \{\replacement\}
\}\{\tl\ var\}
\}

Replaces the earliest match of the regular expression \(?1\regex\ldots\regex\) in the \langle token list variable \rangle by the \langle replacement \rangle corresponding to which \langle regex \rangle matched. If none of the \langle regex \rangle match, then the \langle tl \ var \rangle is not modified. Each \langle regex \rangle can either be given as a regex variable or as an explicit regular expression.

In detail, for each starting position in the \langle token list \rangle, each of the \langle regex \rangle is searched in turn. If one of them matches then it is replaced by the corresponding \langle replacement \rangle as described for \regexReplaceCaseOnce. This is equivalent to checking with \regexMatchCase which \langle regex \rangle matches, then performing the replacement with \regexReplaceCaseOnce.

\regexReplaceCaseOnceT
\{
   \{\regex\} \{\replacement\}
   \{\regex\} \{\replacement\}
   \ldots
   \{\regex\} \{\replacement\}
\}\{\tl\ var\}
\}\{\true\ code\}

Replaces the earliest match of the regular expression \(?1\regex\ldots\regex\) in the \langle token list variable \rangle by the \langle replacement \rangle corresponding to which \langle regex \rangle matched, then leaves the \langle true code \rangle in the input stream. If none of the \langle regex \rangle match, then the \langle tl \ var \rangle is not modified. Each \langle regex \rangle can either be given as a regex variable or as an explicit regular expression.

\regexReplaceCaseOnceF
\{
   \{\regex\} \{\replacement\}
   \{\regex\} \{\replacement\}
   \ldots
   \{\regex\} \{\replacement\}
\}\{\tl\ var\}
\}\{\false\ code\}

Replaces the earliest match of the regular expression \(?1\regex\ldots\regex\) in the \langle token list variable \rangle by the \langle replacement \rangle corresponding to which \langle regex \rangle matched. If none of the \langle regex \rangle match, then the \langle tl \ var \rangle is not modified, and the \langle false code \rangle is left in the input stream. Each \langle regex \rangle can either be given as a regex variable or as an explicit regular expression.

\regexReplaceCaseOnceTF
\{
   \{\regex\} \{\replacement\}
   \{\regex\} \{\replacement\}
   \ldots
   \{\regex\} \{\replacement\}
\}\{\tl\ var\}
\}\{\true\ code\}\{\false\ code\}

Replaces the earliest match of the regular expression \(?1\regex\ldots\regex\) in the \langle token list variable \rangle by the \langle replacement \rangle corresponding to which \langle regex \rangle matched, then leaves the \langle true code \rangle in the input stream. If none of the \langle regex \rangle match, then the \langle tl \ var \rangle is not modified, and the \langle false code \rangle is left in the input stream. Each \langle regex \rangle can either be given as a regex variable or as an explicit regular expression.
\regexReplaceCaseAll
{
  {@(regex_1)} {@replacement_1}
  @{regex_2} {@replacement_2}
  ...
  @{regex_n} {@replacement_n}
} @{tl var}

Replaces all occurrences of all @{regex} in the @{token list} by the corresponding @{replacement}. Every
match is treated independently, and matches cannot overlap. The result is assigned locally to @{tl var}.

In detail, for each starting position in the @{token list}, each of the @{regex} is searched in turn. If one of
them matches then it is replaced by the corresponding @{replacement}, and the search resumes at the
position that follows this match (and replacement). For instance

\tlSet \lTmpaTl {Hello, world!}
\regexReplaceCaseAll
  {[@{A-Za-z}]+} {``\0''}
  {\b} {---}
  {.} {[\0]}
} \lTmpaTl

results in \lTmpaTl having the contents `Hello'---[,][,]---world'---[!]. Note in particular that
the word-boundary assertion \b did not match at the start of words because the case [A-Za-z]+ matched
at these positions. To change this, one could simply swap the order of the two cases in the argument of
\regexReplaceCaseAll.

\regexReplaceCaseAllT
{
  @{regex_1} {replacement_1}
  @{regex_2} {replacement_2}
  ...
  @{regex_n} {replacement_n}
} @{tl var}
  {@true code}

Replaces all occurrences of all @{regex} in the @{token list} by the corresponding @{replacement}. Every
match is treated independently, and matches cannot overlap. The result is assigned locally to @{tl var},
and the @{true code} is left in the input stream if any replacement was made.

\regexReplaceCaseAllF
{
  @{regex_1} {replacement_1}
  @{regex_2} {replacement_2}
  ...
  @{regex_n} {replacement_n}
} @{tl var}
  {@false code}

Replaces all occurrences of all @{regex} in the @{token list} by the corresponding @{replacement}. Every
match is treated independently, and matches cannot overlap. The result is assigned locally to @{tl var},
and the @{false code} is left in the input stream if not any replacement was made.
Replaces all occurrences of all \(\text{regex}\) in the \(\langle\text{token list}\rangle\) by the corresponding \(\langle\text{replacement}\rangle\). Every match is treated independently, and matches cannot overlap. The result is assigned locally to \(\langle\text{tl var}\rangle\), and the \(\langle\text{true code}\rangle\) or \(\langle\text{false code}\rangle\) is left in the input stream depending on whether any replacement was made or not.

13.5 Syntax of Regular Expressions

13.5.1 Regular Expression Examples

We start with a few examples, and encourage the reader to apply \\regexShow\ to these regular expressions.

- \textbf{Cat} matches the word “Cat” capitalized in this way, but also matches the beginning of the word “Cattle”: use \texttt{\bCat\b} to match a complete word only.

- \texttt{[abc]} matches one letter among “a”, “b”, “c”; the pattern \(\texttt{(a|b|c)}\) matches the same three possible letters (but see the discussion of submatches below).

- \texttt{[A-Za-z]*} matches any number (due to the quantifier *) of Latin letters (not accented).

- \texttt{\c{[A-Za-z]*}} matches a control sequence made of Latin letters.

- \texttt{\_\([\-\_]\)*\_} matches an underscore, any number of characters other than underscore, and another underscore; it is equivalent to \(\_\.[\-\_]\) where . matches arbitrary characters and the lazy quantifier \(\{\}\) means to match as few characters as possible, thus avoiding matching underscores.

- \texttt{[\+\-\]*\d+} matches an explicit integer with at most one sign.

- \texttt{[\+\-\]*\d+\_\{\d+\}\_} matches an explicit integer with any number of + and – signs, with spaces allowed except within the mantissa, and surrounded by spaces.

- \texttt{[\+\-\]*\d+\_\{\d+\}\_} matches an explicit integer or decimal number; using \(\{\_\}\) instead of \(\_\) would allow the comma as a decimal marker.

- \texttt{[\+\-\]*\d+\_\{\d+\}\_\{\d+\}\_\{\d+\}e\{\d+\}\_\} matches an explicit dimension with any unit that \TeX\ knows, where \(\{\}\) means to treat lowercase and uppercase letters identically.

- \texttt{[\+\-\]*\d+\_\{\d+\}\_\{\d+\}e\{\d+\}\_\} matches an explicit floating point number or the special values \texttt{nan} and \texttt{inf} (with signs and spaces allowed).

- \texttt{[\+\-\]*\d+\_\{\d+\}c\_\} matches an explicit integer or control sequence (without checking whether it is an integer variable).

- \texttt{\G\_\{\G\} at the beginning of a regular expression matches and discards (due to \texttt{\K}) everything between the end of the previous match \(\langle\G\rangle\) and what is matched by the rest of the regular expression; this is useful in \regexReplaceAll\ when the goal is to extract matches or submatches in a finer way than with \regexExtractAll.}

While it is impossible for a regular expression to match only integer expressions, \texttt{[\+\-\]*\d+\_\{\d+\}e\{\d+\}\_\}} matches among other things all valid integer expressions (made only with explicit integers). One should follow it with further testing.
13.5.2 Characters in Regular Expressions

Most characters match exactly themselves, with an arbitrary category code. Some characters are special and must be escaped with a backslash (e.g., \* matches a star character). Some escape sequences of the form backslash-letter also have a special meaning (for instance \d matches any digit). As a rule,

- every alphanumeric character (A–Z, a–z, 0–9) matches exactly itself, and should not be escaped, because \A, \B, ... have special meanings;
- non-alphanumeric printable ASCII characters can (and should) always be escaped: many of them have special meanings (e.g., use \(, \), \?, \.
- spaces should always be escaped (even in character classes);
- any other character may be escaped or not, without any effect: both versions match exactly that character.

Note that these rules play nicely with the fact that many non-alphanumeric characters are difficult to input into \TeX under normal category codes. For instance, \abc% matches the characters \abc% (with arbitrary category codes), but does not match the control sequence \abc followed by a percent character. Matching control sequences can be done using the \c{⟨regex⟩} syntax (see below).

Any special character which appears at a place where its special behaviour cannot apply matches itself instead (for instance, a quantifier appearing at the beginning of a string), after raising a warning.

Characters.

\texttt{\x{hh…}} Character with hex code hh…

\texttt{\xhh} Character with hex code hh.

\texttt{\a} Alarm (hex 07).

\texttt{\e} Escape (hex 1B).

\texttt{\f} Form-feed (hex 0C).

\texttt{\n} New line (hex 0A).

\texttt{\r} Carriage return (hex 0D).

\texttt{\t} Horizontal tab (hex 09).

13.5.3 Characters Classes

Character types.

. A single period matches any token.

\texttt{\d} Any decimal digit.

\texttt{\h} Any horizontal space character, equivalent to [\ \^I]: space and tab.

\texttt{\s} Any space character, equivalent to [\ \^I\^J\^L\^M].

\texttt{\v} Any vertical space character, equivalent to [\^J\^K\^L\^M]. Note that \^K is a vertical space, but not a space, for compatibility with Perl.

\texttt{\w} Any word character, i.e., alphanumerics and underscore, equivalent to the explicit class [A–Za–z0–9\_].

\texttt{\D} Any token not matched by \d.

\texttt{\H} Any token not matched by \h.

\texttt{\N} Any token other than the \n character (hex 0A).
\S Any token not matched by \s.
\V Any token not matched by \v.
\W Any token not matched by \w.

Of those, ., \d, \h, \n, \s, \v, and \w match arbitrary control sequences.
Character classes match exactly one token in the subject.

[...] Positive character class. Matches any of the specified tokens.
[^...] Negative character class. Matches any token other than the specified characters.
x-y Within a character class, this denotes a range (can be used with escaped characters).

[:(name):] Within a character class (one more set of brackets), this denotes the POSIX character class ⟨name⟩,
which can be alnum, alpha, ascii, blank, cntrl, digit, graph, lower, print, punct, space, upper, word, or xdigit.
[:^⟨name⟩:] Negative POSIX character class.

For instance, [a-oq-z\cC.] matches any lowercase latin letter except p, as well as control sequences (see below for a description of \c).

In character classes, only [, ^, -, ], \ and spaces are special, and should be escaped. Other non-alphanumeric characters can still be escaped without harm. Any escape sequence which matches a single character (\d, \D, etc.) is supported in character classes. If the first character is ^, then the meaning of the character class is inverted; ^ appearing anywhere else in the range is not special. If the first character (possibly following a leading ^) is ] then it does not need to be escaped since ending the range there would make it empty. Ranges of characters can be expressed using ^, for instance, [\D 0-5] and [^6-9] are equivalent.

13.5.4 Structure: Alternatives, Groups, Repetitions

Quantifiers (repetition).

? 0 or 1, greedy.
?? 0 or 1, lazy.

* 0 or more, greedy.
*? 0 or more, lazy.

+ 1 or more, greedy.
+? 1 or more, lazy.

{n} Exactly n.

{n,} n or more, greedy.

{n,}? n or more, lazy.

{n,m} At least n, no more than m, greedy.

{n,m}? At least n, no more than m, lazy.

For greedy quantifiers the regex code will first investigate matches that involve as many repetitions as possible, while for lazy quantifiers it investigates matches with as few repetitions as possible first.

Alternation and capturing groups.

A|B|C Either one of A, B, or C, investigating A first.
Capturing groups.

Non-capturing group.

Non-capturing group which resets the group number for capturing groups in each alternative. The following group is numbered with the first unused group number.

Capturing groups are a means of extracting information about the match. Parenthesized groups are labelled in the order of their opening parenthesis, starting at 1. The contents of those groups corresponding to the "best" match (leftmost longest) can be extracted and stored in a sequence of token lists using for instance \regexExtractOnceTF.

The \K escape sequence resets the beginning of the match to the current position in the token list. This only affects what is reported as the full match. For instance,

\regexExtractAll {a \K .} {a123aaxyz} \lFooSeq

results in \lFooSeq containing the items \{1\} and \{a\}: the true matches are \{a1\} and \{aa\}, but they are trimmed by the use of \K. The \K command does not affect capturing groups: for instance,

\regexExtractOnce {(. \K c)+ \d} {acbc3} \lFooSeq

results in \lFooSeq containing the items \{c3\} and \{bc\}: the true match is \{acbc3\}, with first submatch \{bc\}, but \K resets the beginning of the match to the last position where it appears.

### 13.5.5 Matching Exact Tokens

The \c escape sequence allows to test the category code of tokens, and match control sequences. Each character category is represented by a single uppercase letter:

- \cC for control sequences;
- \cB for begin-group tokens;
- \cE for end-group tokens;
- \cM for math shift;
- \cT for alignment tab tokens;
- \cP for macro parameter tokens;
- \cU for superscript tokens (up);
- \cD for subscript tokens (down);
- \cS for spaces;
- \cL for letters;
- \cO for others; and
- \cA for active characters.

The \c escape sequence is used as follows.

\c{\langle regex\rangle} A control sequence whose csname matches the \langle regex\rangle, anchored at the beginning and end, so that \c{\begin} matches exactly \begin, and nothing else.

\cX Applies to the next object, which can be a character, escape character sequence such as \x{0A}, character class, or group, and forces this object to only match tokens with category X (any of CBEMTPUDSLOA). For instance, \cL[A-Z\d] matches uppercase letters and digits of category code letter, \cC. matches any control sequence, and \cO(abc) matches abc where each character has category other.\footnote{This last example also captures “abc” as a regex group; to avoid this use a non-capturing group \cO{?:abc}.}
\c[XYZ] Applies to the next object, and forces it to only match tokens with category X, Y, or Z (each being any of \texttt{CBEMTPUDSLOA}). For instance, \c[LSO](..) matches two tokens of category letter, space, or other.

\c[\-XYZ] Applies to the next object and prevents it from matching any token with category X, Y, or Z (each being any of \texttt{CBEMTPUDSLOA}). For instance, \c[\-Q]d matches digits which have any category different from other.

The category code tests can be used inside classes; for instance, [\c0\d \c[LSO][A-F]] matches what \TeX{} considers as hexadecimal digits, namely digits with category other, or uppercase letters from A to F with category either letter or other. Within a group affected by a category code test, the outer test can be overridden by a nested test: for instance, \cL(ab\c[CD]\*cd) matches \texttt{ab*cd} where all characters are of category letter, except * which has category other.

The \texttt{\u} escape sequence allows to insert the contents of a token list directly into a regular expression or a replacement, avoiding the need to escape special characters. Namely, \texttt{\u{⟨var name⟩}} matches the exact contents (both character codes and category codes) of the variable \texttt{⟨var name⟩}. Within a \c{...} control sequence matching, the \texttt{\u} escape sequence only expands its argument once. Quantifiers are supported.

The \texttt{\ur} escape sequence allows to insert the contents of a \texttt{regex} variable into a larger regular expression. For instance, \texttt{A\ur{lTmpaRegex}D} matches the tokens A and D separated by something that matches the regular expression \texttt{lTmpaRegex}. This behaves as if a non-capturing group were surrounding \texttt{lTmpaRegex}, and any group contained in \texttt{lTmpaRegex} is converted to a non-capturing group. Quantifiers are supported.

For instance, if \texttt{lTmpaRegex} has value \texttt{B|C}, then \texttt{A\ur{1\_tmpa\_regex}D} is equivalent to \texttt{A(?:\B|\C)D} (matching \texttt{ABD} or \texttt{ACD}) and not to \texttt{AB|CD} (matching \texttt{AB} or \texttt{CD}). To get the latter effect, it is simplest to use \TeX{}’s expansion machinery directly: if \texttt{lTmpaTl} contains \texttt{B|C} then the following two lines show the same result:

\begin{verbatim}
\regexShow {A \u{lTmpaTl} D}
\regexShow {A B | C D}
\end{verbatim}

13.5.6 Miscellaneous

Anchors and simple assertions.

\b Word boundary: either the previous token is matched by \w and the next by \W, or the opposite. For this purpose, the ends of the token list are considered as \W.

\B Not a word boundary: between two \w tokens or two \W tokens (including the boundary).

\^ or \A Start of the subject token list.

\$ \, \Z or \z End of the subject token list.

\G Start of the current match. This is only different from \^ in the case of multiple matches: for instance \texttt{\regexCount {\G a} {aaba} lTmpaInt} yields 2, but replacing \G by \^ would result in \texttt{lTmpaInt} holding the value 1.

The option (?!) makes the match case insensitive (identifying A–Z with a–z; no Unicode support yet). This applies until the end of the group in which it appears, and can be reverted using (?!). For instance, in (?!)(a(?!b)c)d, the letters a and d are affected by the i option. Characters within ranges and classes are affected individually: (?!)[Y-\]\ is equivalent to [YZ]\[yz], and (?!)[^aeiou] matches any character which is not a vowel. Neither character properties, nor \c{...} nor \u{...} are affected by the i option.
13.6 Syntax of the Replacement Text

Most of the features described in regular expressions do not make sense within the replacement text. Backslash introduces various special constructions, described further below:

- \0 is the whole match;
- \1 is the submatch that was matched by the first (capturing) group (\ldots); similarly for \2, \ldots, \9 and \g{⟨number⟩};
- \w inserts a space (spaces are ignored when not escaped);
- \a, \e, \f, \n, \r, \t, \xhh, \x{hhh} correspond to single characters as in regular expressions;
- \c{⟨cs name⟩} inserts a control sequence;
- \c{⟨category⟩⟨character⟩} (see below);
- \u{⟨tl var name⟩} inserts the contents of the ⟨tl var⟩ (see below).

Characters other than backslash and space are simply inserted in the result (but since the replacement text is first converted to a string, one should also escape characters that are special for \TeX, for instance use \#). Non-alphanumeric characters can always be safely escaped with a backslash.

For instance,

\tlSet \lTmpaTl {Hello, world!}
\regexReplaceAll {([er]?l|o) .} {\{\0--\1\}} \lTmpaTl
\tlUse \lTmpaTl

H(ell–el)(o,–o) w(or–o)(ld–l)!

The submatches are numbered according to the order in which the opening parenthesis of capturing groups appear in the regular expression to match. The \n-th submatch is empty if there are fewer than \n capturing groups or for capturing groups that appear in alternatives that were not used for the match. In case a capturing group matches several times during a match (due to quantifiers) only the last match is used in the replacement text. Submatches always keep the same category codes as in the original token list.

By default, the category code of characters inserted by the replacement are determined by the prevailing category code regime at the time where the replacement is made, with two exceptions:

- space characters (with character code 32) inserted with \w or \x20 or \x{20} have category code 10 regardless of the prevailing category code regime;
- if the category code would be 0 (escape), 5 (newline), 9 (ignore), 14 (comment) or 15 (invalid), it is replaced by 12 (other) instead.

The escape sequence \c allows to insert characters with arbitrary category codes, as well as control sequences.

\cX(⟨...⟩) Produces the characters “⟨...⟩” with category \X, which must be one of CBEMTPUDSLOA as in regular expressions. Parentheses are optional for a single character (which can be an escape sequence). When nested, the innermost category code applies, for instance \cL(Hello\cS\ world)! gives this text with standard category codes.

\c{⟨text⟩} Produces the control sequence with csname ⟨text⟩. The ⟨text⟩ may contain references to the submatches \0, \1, and so on, as in the example for \u below.

The escape sequence \u{⟨var name⟩} allows to insert the contents of the variable with name ⟨var name⟩ directly into the replacement, giving an easier control of category codes. When nested in \c{⟨...⟩} and \u{⟨...⟩} constructions, the \u and \c escape sequences extract the value of the control sequence and turn it into a string. Matches can also be used within the arguments of \c and \u. For instance,
Regex replacement is also a convenient way to produce token lists with arbitrary category codes. For instance

```markdown
\tlClear \lTmpaTl
\regexReplaceAll { } {\cU \% \cA \~} \lTmpaTl
```

results in `\lTmpaTl` containing the percent character with category code 7 (superscript) and an active tilde character.
Chapter 14

Token Manipulation (Token)

\charLowercase \langle \text{char} \rangle
\charUppercase \langle \text{char} \rangle
\charTitlecase \langle \text{char} \rangle
\charFoldcase \langle \text{char} \rangle

Converts the \langle char \rangle to the equivalent case-changed character as detailed by the function name (see \text{\textTitlecase} for details of these terms). The case mapping is carried out with no context-dependence (cf. \text{\textUppercase}, etc.) These functions generate characters with the category code of the \langle char \rangle (i.e. only the character code changes).

\charStrLowercase \langle \text{char} \rangle
\charStrUppercase \langle \text{char} \rangle
\charStrTitlecase \langle \text{char} \rangle
\charStrFoldcase \langle \text{char} \rangle

Converts the \langle char \rangle to the equivalent case-changed character as detailed by the function name (see \text{\textTitlecase} for details of these terms). The case mapping is carried out with no context-dependence (cf. \text{\textUppercase}, etc.) These functions generate “other” (category code 12) characters.

\charSetLccode \{\langle intexpr_1 \rangle\} \{\langle intexpr_2 \rangle\}

Sets up the behaviour of the \langle character \rangle when found inside \text{\textLowercase}, such that \langle character_1 \rangle will be converted into \langle character_2 \rangle. The two \langle characters \rangle may be specified using an \langle integer expression \rangle for the character code concerned. This may include the \TeX \langle character \rangle method for converting a single character into its character code:

\begin{verbatim}
\charSetLccode {\`A} {\`a} \% Standard behaviour
\charSetLccode {\`A} {\`A + 32}
\charSetLccode {65} {97}
\end{verbatim}

The setting applies within the current \TeX group.

\charSetUccode \{\langle intexpr_1 \rangle\} \{\langle intexpr_2 \rangle\}

Sets up the behaviour of the \langle character \rangle when found inside \text{\textUppercase}, such that \langle character_1 \rangle will be converted into \langle character_2 \rangle. The two \langle characters \rangle may be specified using an \langle integer expression \rangle for the character code concerned. This may include the \TeX \langle character \rangle method for converting a single character into its character code:
\charSetUccode \textasciitilde \textasciitilde \% Standard behaviour
\charSetUccode \textasciitilde \textasciitilde \texttilde \textasciitilde \textasciitilde - 32
\charSetUccode 97 \{65\}

The setting applies within the current \TeX\ group.

\charValueLccode \langle integer expression\rangle

Returns the current lower case code of the \langle character\rangle with character code given by the \langle integer expression\rangle.

\charValueUccode \langle integer expression\rangle

Returns the current upper case code of the \langle character\rangle with character code given by the \langle integer expression\rangle.
Chapter 15

Text Processing (Text)

This module deals with manipulation of (formatted) text; such material is comprised of a restricted set of token list content. The functions provided here concern conversion of textual content for example in case changing, Begin-group and end-group tokens in the ⟨text⟩ are normalized and become { and }, respectively.

15.1 Case Changing

These case changing functions are designed to work with Unicode input only. As such, UTF-8 input is assumed for all engines. When used with XeTeX or LuaTeX a full range of Unicode transformations are enabled. Specifically, the standard mappings here follow those defined by the Unicode Consortium in UnicodeData.txt and SpecialCasing.txt. In the case of 8-bit engines, mappings are provided for characters which can be represented in output typeset using the T1, T2 and LGR font encodings. Thus for example ä can be case-changed using pdfTeX. For pTeX only the ASCII range is covered as the engine treats input outside of this range as east Asian.

\textExpand {⟨text⟩}

Takes user input ⟨text⟩ and expands the content. Protected commands (typically formatting) are left in place, and no processing takes place of math mode material. Commands which are neither engine- nor \LaTeX protected are expanded exhaustively.

\textLowercase {⟨tokens⟩}
\textUppercase {⟨tokens⟩}
\textTitlecase {⟨tokens⟩}
\textTitlecaseFirst {⟨tokens⟩}

Takes user input ⟨text⟩ first applies \textExpand, then transforms the case of character tokens as specified by the function name. The category code of letters are not changed by this process (at least where they can be represented by the engine as a single token: 8-bit engines may require active characters).

Upper- and lowercase have the obvious meanings. Titlecasing may be regarded informally as converting the first character of the ⟨tokens⟩ to uppercase and the rest to lowercase. However, the process is more complex than this as there are some situations where a single lowercase character maps to a special form, for example ij in Dutch which becomes IJ.

For titlecasing, note that there are two functions available. The function \textTitlecase applies (broadly) uppercasing to the first letter of the input, then lowercasing to the remainder. In contrast, \textTitlecaseFirst only carries out the uppercasing operation, and leaves the balance of the input unchanged.

Case changing does not take place within math mode material. For example:

\textUppercase {Text $y=mx+c$ with {Braces}}

\begin{Verbatim}
TEXT $y = mx + c$ WITH BRACES
\end{Verbatim}
Takes user input \textit{(text)} first applies \texttt{textExpand}, then transforms the case of character tokens as specified by the function name. The category code of letters are not changed by this process (at least where they can be represented by the engine as a single token: 8-bit engines may require active characters).

These conversions are language-sensitive, and follow Unicode Consortium guidelines. Currently, the languages recognised for special handling are as follows.

- Azeri and Turkish (\texttt{az} and \texttt{tr}). The case pairs i/dotless and I-dot/i are activated for these languages. The combining dot mark is removed when lowercasing I-dot and introduced when upper casing i-dotless.

- German (\texttt{de-alt}). An alternative mapping for German in which the lowercase Eszett maps to a \textit{großes Eszett}. Since there is a T1 slot for the \textit{großes Eszett} in T1, this tailoring \textit{is} available with pdfTeX as well as in the Unicode \TeX engines.

- Greek (\texttt{el}). Removes accents from Greek letters when uppercasing; titlecasing leaves accents in place. (At present this is implemented only for Unicode engines.)

- Lithuanian (\texttt{lt}). The lowercase letters i and j should retain a dot above when the accents grave, acute or tilde are present. This is implemented for lowercasing of the relevant uppercase letters both when input as single Unicode codepoints and when using combining accents. The combining dot is removed when uppercasing in these cases. \textit{Note that only} the accents used in Lithuanian are covered: the behaviour of other accents are not modified.

- Dutch (\texttt{nl}). Capitalisation of ij at the beginning of titlecased input produces IJ rather than Ij. The output retains two separate letters, thus this transformation \textit{is} available using pdfTeX.
Chapter 16

Files (File)

This module provides functions for working with external files.

It is important to remember that when reading external files \TeX{} attempts to locate them using both the operating system path and entries in the \TeX{} file database (most \TeX{} systems use such a database). Thus the “current path” for \TeX{} is somewhat broader than that for other programs.

For functions which expect a \langle file name \rangle argument, this argument may contain both literal items and expandable content, which should on full expansion be the desired file name. Quote tokens (") are not permitted in file names as they are reserved for internal use by some \TeX{} primitives.

Spaces are trimmed at the beginning and end of the file name: this reflects the fact that some file systems do not allow or interact unpredictably with spaces in these positions. When no extension is given, this will trim spaces from the start of the name only.

16.1 File Operation Functions

\fileInput {⟨file name⟩}

Searches for \langle file name \rangle in the path as detailed for \fileIfExistTF, and if found reads in the file and returns the contents. All files read are recorded for information and the file name stack is updated by this function. An error is raised if the file is not found.

\fileIfExistInput {⟨file name⟩}
\fileIfExistInputF {⟨file name⟩} {⟨false code⟩}

Searches for \langle file name \rangle using the current \TeX{} search path. If found then reads in the file and returns the contents as described for \fileInput, otherwise inserts the \langle false code \rangle. Note that these functions do not raise an error if the file is not found, in contrast to \fileInput.

\fileGet {⟨filename⟩} {⟨setup⟩} {⟨tl⟩}
\fileGetT {⟨filename⟩} {⟨setup⟩} {⟨tl⟩} {⟨true code⟩}
\fileGetF {⟨filename⟩} {⟨setup⟩} {⟨tl⟩} {⟨false code⟩}
\fileGetTF {⟨filename⟩} {⟨setup⟩} {⟨tl⟩} {⟨true code⟩} {⟨false code⟩}

Defines \langle tl \rangle to the contents of \langle filename \rangle. Category codes may need to be set appropriately via the \langle setup \rangle argument. The non-branching version sets the \langle tl \rangle to \qNoValue if the file is not found. The branching version runs the \langle true code \rangle after the assignment to \langle tl \rangle if the file is found, and \langle false code \rangle otherwise.
\fileIfExist {⟨file name⟩}
\fileIfExistT {⟨file name⟩} {⟨true code⟩}
\fileIfExistF {⟨file name⟩} {⟨false code⟩}
\fileIfExistTF {⟨file name⟩} {⟨true code⟩} {⟨false code⟩}

Searches for ⟨file name⟩ using the current \TeX{} search path.
Chapter 17

Quarks (Quark)

Quarks are control sequences (and in fact, token lists) that expand to themselves and should therefore never be executed directly in the code. This would result in an endless loop!

Quarks can be used as error return values for functions that receive erroneous input. For example, in the function \propGet to retrieve a value stored in some key of a property list, if the key does not exist then the return value is the quark \qNoValue. As mentioned above, such quarks are extremely fragile and it is imperative when using such functions that code is carefully written to check for pathological cases to avoid leakage of a quark into an uncontrolled environment.

17.1 Constant Quarks

\qNoValue

A canonical value for a missing value, when one is requested from a data structure. This is therefore used as a “return” value by functions such as \propGet if there is no data to return.

17.2 Quark Conditionals

\quarkVarIfNoValue (token)
\quarkVarIfNoValueT (token) {⟨true code⟩}
\quarkVarIfNoValueF (token) {⟨false code⟩}
\quarkVarIfNoValueTF (token) {⟨true code⟩} {⟨false code⟩}

Tests if the ⟨token⟩ is equal to \qNoValue.

\clistGet \cEmptyClist \lTmpaTl
\quarkVarIfNoValueTF \lTmpaTl {\prgReturn{NoValue}} {\prgReturn{SomeValue}}

\seqPop \cEmptySeq \lTmpaTl
\quarkVarIfNoValueTF \lTmpaTl {\prgReturn{NoValue}} {\prgReturn{SomeValue}}

\propSetFromKeyval \lTmpaProp {key1=one,key2=two}
\propGet \lTmpaProp {key3} \lTmpaTl
\quarkVarIfNoValueTF \lTmpaTl {\prgReturn{NoValue}} {\prgReturn{SomeValue}}

108
Chapter 18

Legacy Concepts (Legacy)

There are a small number of \TeX\ or \LaTeX\ concepts which are not used in functional code but which need to be manipulated when working as a \LaTeX\ package. To allow these to be integrated cleanly into functional code, a set of legacy interfaces are provided here.

\begin{align*}
\text{\texttt{\LaTeXIf\{}{(name)}\}\texttt{\LaTeXIfT\{}{(true\ code)}\}\texttt{\LaTeXIfF\{}{(false\ code)}\}\texttt{\LaTeXIfTF\{}{(name)}\ {(true\ code)}\ {(false\ code)}\}
\end{align*}

Tests if the \LaTeX\ conditional (generated by \texttt{\newif}) if true or false and branches accordingly. The \langle name\rangle of the conditional should omit the leading if.

\begin{verbatim}
\newif \ifFooBar
\legacyIfTF {FooBar} \prgReturn{True!} \prgReturn{False!}
\end{verbatim}

\begin{verbatim}
\newif \ifFooBar
\legacyIfSetTrue {FooBar}
\legacyIfTF {FooBar} \prgReturn{True!} \prgReturn{False!}
\end{verbatim}

\begin{verbatim}
\newif \ifFooBar
\legacyIfSet {FooBar} {cFalseBool}
\legacyIfTF {FooBar} \prgReturn{True!} \prgReturn{False!}
\end{verbatim}
Chapter 19

The Source Code

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

\RequirePackage{expl3}
\ProvidesExplPackage{functional}{2024-02-16}{2024A}
{^^JIntuitive Functional Programming Interface for LaTeX2}
\cs_generate_variant:Nn \iow_log:n { V }
\cs_generate_variant:Nn \str_set:Nn { Ne }
\cs_generate_variant:Nn \tl_log:n { e }
\cs_generate_variant:Nn \tl_set:Nn { Ne }
\prg_generate_conditional_variant:Nnn \str_if_eq:nn { Ve } { TF }
\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

19.1 Interfaces for Functional Programming (Prg)

19.1.1 Setting Functional Package

\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_off:e #1 { }
\cs_new_protected:Npn \__fun_tracing_log_off:V #1 { }

\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 }

19.1.2 Creating New Functions and Conditionals

\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__tl     {  } % 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} }
\tl_const:Nn \c__fun_parameter_called_i_ii_tl  { {#1}{#2} }
\tl_const:Nn \c__fun_parameter_called_i_iii_tl { {#1}{#2}{#3} }
\tl_const:Nn \c__fun_parameter_called_i_iv_tl  { {#1}{#2}{#3}{#4} }
\tl_const:Nn \c__fun_parameter_called_i_v_tl   { {#1}{#2}{#3}{#4}{#5} }
\tl_const:Nn \c__fun_parameter_called_i_vi_tl  { {#1}{#2}{#3}{#4}{#5}{#6} }
\tl_const:Nn \c__fun_parameter_called_i_vii_tl { {#1}{#2}{#3}{#4}{#5}{#6}{#7} }
\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 }
\_\_fun\_parameters\_defined\_tl
{
\_\_fun\_group\_begin:
\tl_gclear:N \gResultTl
#3
\_\_fun\_tracing\_log:e { [0] - \exp_not:V \gResultTl }
\_\_fun\_group\_end:
\use:c \{ __fun\_new\_with\_arg\_ \int_to_roman:n \{ \_\_fun\_arg\_count\_int \} :NnV \}
#1 {#2} \_\_fun\_parameters\_defined\_tl
}
\cs_generate_variant:Nn \__fun\_new\_function:Nnn { cne }
\cs_set_eq:NN \prgNewFunction \__fun\_new\_function:Nnn
\cs_set_eq:NN \PrgNewFunction \__fun\_new\_function:Nnn
\tl_new:N \g__fun\_last\_result\_tl
\int_new:N \l__fun\_cond\_arg\_count\_int
%% #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:Nn \l__fun\_cond\_arg\_count\_int { \tl_count:n {#2} }
\tl_set_eq:Nc \l__fun\_parameters\_called\_tl
{
\c__fun\_parameter\_called_i_
\int_to_roman:n \{ \l__fun\_cond\_arg\_count\_int \}_tl
}
%% define function \FooIfBarT for #1=\FooIfBar
\tl_set_eq:Nc \l__fun\_parameters\_true\_tl
{
\c__fun\_parameter\_called_
\int_to_roman:n \{ \l__fun\_cond\_arg\_count\_int + 1 \}_tl
}
\_\_fun\_new\_function:cne \{ \cs_to_str:N \#1 \_T \} \{ \#2 \_n \}
{
#1 \exp_not:V \_\_fun\_parameters\_called\_tl
\exp_not:n
{
\tl_set_eq:NN \g__fun\_last\_result\_tl \gResultTl
\tl_gclear:N \gResultTl
\exp_last_unbraced:NV \bool_if:NT \g__fun\_last\_result\_tl
\exp_not:V \_\_fun\_parameters\_true\_tl
}
%% define function \FooIfBarF for #1=\FooIfBar
\tl_set_eq:Nc \l__fun\_parameters\_false\_tl
{
\c__fun\_parameter\_called_
\int_to_roman:n \{ \l__fun\_cond\_arg\_count\_int + 1 \}_tl
}
\_\_fun\_new\_function:cne \{ \cs_to_str:N \#1 \_F \} \{ \#2 \_n \}
{
#1 \exp_not:V \_\_fun\_parameters\_called\_tl
\exp_not:n
{
```latex
\texttt{\tl_set_eq:NN \g__fun_last_result_tl \gResultTl}
\texttt{\tl_gclear:N \gResultTl}
\texttt{\exp_last_unbraced:NV \bool_if:NF \g__fun_last_result_tl}
\texttt{\exp_not:V \l__fun_parameters_false_tl}
\texttt{\%\% define function \FooIfBarTF for \#1=\FooIfBar}
\texttt{\tl_set_eq:Nc \l__fun_parameters_true_tl}
\texttt{\c__fun_parameter_called_}
\texttt{\int_to_roman:n { \l__fun_cond_arg_count_int + 1 } _tl}
\texttt{\tl_set_eq:Nc \l__fun_parameters_false_tl}
\texttt{\c__fun_parameter_called_}
\texttt{\int_to_roman:n { \l__fun_cond_arg_count_int + 2 } _tl}
\texttt{\__fun_new_function:cne { \cs_to_str:N #1 TF } { #2 n n }}
\texttt{\#1 \exp_not:V \l__fun_parameters-called_tl}
\texttt{\exp_not:n}
\texttt{\{ \tl_set_eq:NN \g__fun_last_result_tl \gResultTl}
\texttt{\tl_gclear:N \gResultTl}
\texttt{\exp_last_unbraced:NV \bool_if:NF \g__fun_last_result_tl}
\texttt{\exp_not:V \l__fun_parameters_true_tl}
\texttt{\exp_not:V \l__fun_parameters_false_tl}
\texttt{\}}
\texttt{\}}
\texttt{\cs_set_eq:NN \prgNewConditional \__fun_new_conditional:Nnn}
\texttt{\cs_set_eq:NN \PrgNewConditional \__fun_new_conditional:Nnn}
\texttt{\int_new:N \g__fun_nesting_level_int}
\texttt{\%\% #1: function name; #2: argument specifications; #3 parameters tl defined}
\texttt{\%\% Some times we need to create a function without arguments}
\texttt{\cs_new_protected:Npn \__fun_new_with_arg_:Nnn #1 #2 #3}
\texttt{\{ \cs_new_protected:Npn #1 #3}
\texttt{\{ \int_gincr:N \g__fun_nesting_level_int}
\texttt{\__fun_evaluate:Nn #1 {#2}}
\texttt{\int_gdecr:N \g__fun_nesting_level_int}
\texttt{\__fun_return_result:}
\texttt{\}}
\texttt{\}}
\texttt{\cs_generate_variant:Nn \__fun_new_with_arg_:Nnn { NnV }}
\texttt{\%\% #1: function name; #2: argument specifications; #3 parameters tl defined}
\texttt{\cs_new_protected:Npn \__fun_new_with_arg_i:Nnn #1 #2 #3}
\texttt{\{ \cs_new_protected:Npn #1 #3}
\texttt{\{ \int_gincr:N \g__fun_nesting_level_int}
\texttt{\__fun_one_argument_gset:nn { 1 } { ##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 }
%
\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 }
%
\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_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 { NnV }
%
\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 { NnV }
%
\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 { NnV }
%% #1: function name; #2: argument specifications; #3 parameters tl defined
\cs_new_protected:Npn \__fun_new_with_arg_vi:Nnn #1 #2 #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
\{
\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_e_tl { e }
\tl_const:Nn \c__fun_argtype_E_tl { E }
\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: } \_\_fun_argument_tl
\tl_case:Nn \_\_fun_argtype_tl
 { \c\_\_fun_argtype_e_tl
  { \_\_fun_evaluate_all_and_put_argument:N \l_\_fun_argument_tl }
 }\c\_\_fun_argtype_E_tl
 { \_\_fun_evaluate_all_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_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_all_and_put_argument:N #1
 { \_\_fun_eval_all:V #1
 \_\_fun_arguments_gput:e { \exp_not:V \gResultTl }
 }

\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
   }
  { #1 \_\_fun_arguments_gput:e { \exp_not:V \gResultTl }
   }
  { \_\_fun_arguments_gput:e { \exp_not:V #1 }
   }
  }

% #1: argument number; #2: token lists
\cs_new_protected:Npn \_\_fun_one_argument_gset:nn #1 #2
\{ \\
\tl_gset:cn
  \{ g\_fun_one_argument_ \int_use:N g\_fun_nesting_level_int _#1_tl \} \{ #2 \}
  \^{ g\_fun_one_argument_log:nn \{ #1 \} \{ set \}}
\%
\%
\% #1: argument number; #2: variable of token lists
\cs_new_protected:Npn __fun_one_argument_get:nN #1 #2
\{ \\
  \tl_set_eq:Nc
  #2 { g\_fun_one_argument_ \int_use:N g\_fun_nesting_level_int _#1_tl }
  \^{ __fun_one_argument_log:nn \{ #1 \} \{ get \}}
\}
\cs_generate_variant:Nn __fun_one_argument_get:nN { eN }
\%
\% #1: argument number; #2: get or set
\cs_new_protected:Npn __fun_one_argument_log:nn #1 #2
\{ \\
  \tl_log:e
  \{ #2 \_level _ \int_use:N g\_fun_nesting_level_int _ arg _#1 _ = _ = _ \exp_not:v
  \{ g\_fun_one_argument_ \int_use:N g\_fun_nesting_level_int _#1_tl \}
  \}
\}
\int_new:c \{ g\_fun_argtype_index_ 1 _int \}
\int_new:c \{ g\_fun_argtype_index_ 2 _int \}
\int_new:c \{ g\_fun_argtype_index_ 3 _int \}
\int_new:c \{ g\_fun_argtype_index_ 4 _int \}
\int_new:c \{ g\_fun_argtype_index_ 5 _int \}
\cs_new_protected:Npn __fun_argtype_index_gzero:
\{ \\
  \int_gzero_new:c
  \{ g\_fun_argtype_index_ \int_use:N g\_fun_nesting_level_int _int \}
\}
\cs_new_protected:Npn __fun_argtype_index_gincr:
\{ \\
  \int_gincr:c
  \{ g\_fun_argtype_index_ \int_use:N g\_fun_nesting_level_int _int \}
\}
\cs_new: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 { \_\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 { \_\fun_arguments_ \int_use:N \g__fun_nesting_level_int_tl }
    }
  }

\cs_new_protected:Npn \__fun_arguments_gput:n #1
  {
    \tl_gput_right:cn { \_\fun_arguments_ \int_use:N \g__fun_nesting_level_int_tl } { #1 }
  }
\cs_generate_variant:Nn \__fun_arguments_gput:n { e }

19.1.3 Creating Some Useful Functions

\prgNewFunction \prgSetEqFunction { N N }
  {
    \cs_set_eq:NN #1 #2
    \cs_set_eq:cc \__fun_defined_ \cs_to_str:N #1 : w
    \__fun_defined_ \cs_to_str:N #2 : w
  }

\prgNewFunction \prgDo {n} {#1}
  {
    \prgBreak \prg_break:
    \prgBreakDo \prg_break:n
  }

19.1.4 Return Values and Return Processors

\cs_new_protected:Npn \__fun_put_result:n #1
  {
    \tl_gput_right:Nn \gResultTl { #1 }
  }
\cs_generate_variant:Nn \__fun_put_result:n { V, e, f, o }

\prgNewFunction \prgReturn { m }
  {
    \__fun_put_result:n { #1 }
  }

%% Obsolete function, will be removed in the future
%% We can not define it with \PrgSetEqFunction
\PrgNewFunction \Result { m }
  {
    \__fun_put_result:n { #1 }
  }
\texttt{\textbackslash int\_new:N \_\_fun\_return\_level\_int}

%% By default, the result is returned only if the function is not
%% nested in another function, but this behavior can be customized
\texttt{\textbackslash cs\_new\_protected:Npn \_\_fun\_return\_result:}
\{
  \texttt{\textbackslash int\_compare:NNn \_\_fun\_nesting\_level\_int = \_\_fun\_return\_level\_int}
  \{
    \_\_fun\_use\_result:
  
\}

\texttt{\textbackslash cs\_new\_protected:Npn \_\_fun\_use\_result\_default:}
\{
  \texttt{\tl\_use:N \gResultTl}
\}

%% Set default return processor
\texttt{\textbackslash cs\_new\_protected:Npn \_\_fun\_set\_return\_processor\_default:}
\{
  \texttt{\textbackslash int\_set:Nn \_\_fun\_return\_level\_int 0}
  \texttt{\textbackslash cs\_set\_eq:NN \_\_fun\_use\_result: \_\_fun\_use\_result\_default:}
\}

\_\_fun\_set\_return\_processor\_default:

%% Set current nesting level and return processor
\texttt{\textbackslash cs\_new\_protected:Npn \_\_fun\_set\_return\_processor:n \#1}
\{
  \texttt{\textbackslash int\_set:eq:NN \_\_fun\_return\_level\_int \_\_fun\_nesting\_level\_int}
  \texttt{\textbackslash cs\_set\_protected:Npn \_\_fun\_use\_result: \#1}
\}

%% \#1: return processor; \#2: code to run
%% We do it inside groups for nesting processors to make correct results
\texttt{\textbackslash cs\_new\_protected:Npn \textbackslash fun\_run\_return\_processor:nn \#1 \#2}
\{
  \texttt{\textbackslash group\_begin:}
  \texttt{\_\_fun\_set\_return\_processor:n \#1}
  \texttt{\#2}
  \texttt{\textbackslash group\_end:}
\}

\textbf{19.1.5 Evaluating Functions inside Arguments, I}

%% The function \_\_fun\_eval\_all:n is only used for arguments to be passed
%% to \texttt{\textbackslash int\_eval:n}, \texttt{\textbackslash fp\_eval:n}, \texttt{\textbackslash dim\_eval:n}, and similar functions.
%% It will not keep spaces, and will not distinguish between \{ and \bgroup.
\texttt{\tl\_new:N \_\_fun\_eval\_result\_tl}

%% Evaluate all functions in \#1 and replace them with their return values
\texttt{\textbackslash cs\_new\_protected:Npn \_\_fun\_eval\_all:n \#1}
\{
  \texttt{\textbackslash fun\_run\_return\_processor:nn}
  \{
    \texttt{\exp\_last\_unbraced:NV \_\_fun\_eval\_all\_aux:n \gResultTl}
  \}
\}
19.1.6 Evaluating Functions inside Arguments, II

The function \evalWhole can be used in almost all use cases. It will keep spaces, and will distinguish between { and \bgroup.

\prgNewFunction \evalWhole {n}
{
  \__fun_eval_whole:n {#1}
}

\tl_new:N \l__fun_eval_whole_tl
\bool_new:N \l__fun_eval_none_bool

Evaluate all functions in #1 and replace them with their return values
\cs_new_protected:Npn \__fun_eval_whole:n {#1}
{
  \fun_run_return_processor:nn
  {
    \bool_if:NTF \l__fun_eval_none_bool
    {
      \tl_put_right:Nx \l__fun_eval_whole_tl \exp_not:N \exp_not:n { \exp_not:V \gResultTl }
      \bool_set_false:N \l__fun_eval_none_bool
    }
  }
}

\tl_clear:N \l__fun_eval_result_tl
\__fun_eval_all_aux:n #1 \q_stop
\tl_gset_eq:NN \gResultTl \l__fun_eval_result_tl
}
\cs_generate_variant:Nn \__fun_eval_all:n { V }
\cs_new_protected:Npn \__fun_eval_all_aux:n #1
{
\tl_if_single_token:nTF {#1}
{
  \token_if_eq_meaning:NNF #1 \q_stop
  {
    \bool_lazy_and:nnTF
    { \token_if_cs_p:N #1 }
    { \cs_if_exist_p:c { \__fun_defined_ \cs_to_str:N #1 : w } }
    { #1 }
    {
      \tl_put_right:Nn \l__fun_eval_result_tl {#1}
      \__fun_eval_all_aux:n
    }
  }
}
{
  \%% The braces enclosing a single token (such as {x}) are removed
  \%% but I guess there is no harm inside \int_eval:n or \fp_eval:n
  \tl_put_right:Nn \l__fun_eval_result_tl { {#1} }
  \__fun_eval_all_aux:n
}
}
\cs_new_protected:Npn __fun_eval_whole_aux: 
{
  \peek_analysis_map_inline:n
  \int_compare:nNnTF {##2} = {-1} % control sequence 
  \token_if_eq_meaning:NNTF {##1} \q_stop 
  \cs_if_exist:cTF { __fun_defined_ \exp_last_unbraced:No \cs_to_str:N {##1} : w } 
  \exp_last_unbraced:No \exp_not:N \someFunc 
  \bool_set_true:N \l__fun_eval_none_bool 
  \peek_analysis_map_break:n
  \peek_analysis_map_break: }
  \exp_last_unbraced:No \cs_if_eq:NNT {##1} \evalNone 
  \tl_put_right:Nn \l__fun_eval_whole_tl {##1} 
  \tl_put_right:Nn \l__fun_eval_whole_tl {##1} 
}
\prgNewFunction \evalNone {n} { \tl_gset:Nn \gResultTl {#1} }

% The function \evalNone prevent the evaluation of its argument
\prgNewFunction \prgPrint { m } 
{
  \tl_log:n {running PrgPrint} 
  \int_set_eq:NN \l__fun_return_level_int \g__fun_nesting_level_int 
  #1 
  \int_zero:N \l__fun_return_level_int 
  \tl_gclear:N \gResultTl
}

19.1.7 Printing Contents to the Input Stream

19.1.8 Filling Arguments into Inline Commands

%% To make better tracing log, we want to expand the return value once, but at the same time avoid evaluating the leading function in $\texttt{gResultTl}$, therefore we need to use $\texttt{tl_set:Nn}$ command instead of $\texttt{tlSet}$ function.

\prgNewFunction \prgRunOneArgCode { m n }
{ \cs_set:Npn \__fun_one_arg_cmd:n ##1 {#2} \exp_args:NNo \tl_set:Nn \gResultTl { \__fun_one_arg_cmd:n {#1} } }
\prgNewFunction \prgRunTwoArgCode { m m n }
{ \cs_set:Npn \__fun_two_arg_cmd:nn ##1 ##2 {#3} \exp_args:NNo \tl_set:Nn \gResultTl { \__fun_two_arg_cmd:nn {#1} {#2} } }
\prgNewFunction \prgRunThreeArgCode { m m m n }
{ \cs_set:Npn \__fun_three_arg_cmd:nnn ##1 ##2 ##3 {#4} \exp_args:NNo \tl_set:Nn \gResultTl { \__fun_three_arg_cmd:nnn {#1} {#2} {#3} } }
\prgNewFunction \prgRunFourArgCode { m m m m n }
{ \cs_set:Npn \__fun_four_arg_cmd:nnnn ##1 ##2 ##3 ##4 {#5} \exp_args:NNo \tl_set:Nn \gResultTl { \__fun_four_arg_cmd:nnnn {#1} {#2} {#3} {#4} } }

19.1.9 Checking for Local or Global Variables

\str_new:N \l__fun_variable_name_str
\str_new:N \l__fun_variable_name_a_str
\str_new:N \l__fun_variable_name_b_str

\prg_new_protected_conditional:Nppnn \__fun_if_global_variable:N #1 { TF }
{ \str_set:Nx \l__fun_variable_name_str { \cs_to_str:N #1 } \str_set:Nx \l__fun_variable_name_b_str { \str_item:Nn \l__fun_variable_name_str {2} } \str_if_eq:VTF \l__fun_variable_name_b_str { \str_uppercase:f { \l__fun_variable_name_b_str } }{ \__fun_if_set_local: } }
\bool_new:N \g__fun_variable_local_bool
\cs_new:Npn \__fun_if_set_local:
\{
\bool_if:NTF \g__fun_variable_local_bool
\{
\bool_gset_false:N \g__fun_variable_local_bool
\prg_return_false:
\}
\prg_return_true:
\}
\prgNewFunction \prgLocal { }\{
\bool_gset_true:N \g__fun_variable_local_bool \}
%% 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:
\}

19.2 Interfaces for Argument Using (Use)

\prgNewFunction \expName { m }\{
\exp_args:Nc \__fun_put_result:n { #1 }
\}
\prgNewFunction \expValue { M }\{
\__fun_put_result:V #1
\}
\prgNewFunction \expWhole { m }\{
\__fun_put_result:e { #1 }
\}
\prgNewFunction \expPartial { m }\{
\__fun_put_result:f { #1 }
\}
\prgNewFunction \expOnce { m }\{
\__fun_put_result:o { #1 }
\}
\cs_set_eq:NN \unExpand \exp_not:n
\cs_set_eq:NN \noExpand \exp_not:N
\cs_set_eq:NN \onlyName \exp_not:c
\cs_set_eq:NN \onlyValue \exp_not:V
\cs_set_eq:NN \onlyPartial \exp_not:f
\cs_set_eq:NN \onlyOnce \exp_not:o

\prgNewFunction \useOne { n } { \prgReturn {#1} }
\prgNewFunction \gobbleOne { n } { }
\prgNewFunction \useGobble { n n } { \prgReturn {#1} }
\prgNewFunction \gobbleUse { n n } { \prgReturn {#2} }

19.3 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@FunTmpxBool \bool_new:N \l@FunTmpyBool \bool_new:N \l@FunTmpzBool

\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@FunTmpxBool \bool_new:N \g@FunTmpyBool \bool_new:N \g@FunTmpzBool

\prgNewFunction \boolNew { M } { \bool_new:N #1 }
\prgNewFunction \boolConst { M e } { \bool_const:Nn #1 {#2} }
\prgNewFunction \boolSet { M e } { \__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 #1 { \bool_gset_eq:NN #1 #2 } { \bool_set_eq:NN #1 #2 } }
\prgNewFunction \boolLog { e } { \bool_log:n {#1} }
\prgNewFunction \boolVarLog { M } { \bool_log:N #1 }

\prgNewFunction \boolShow { e } { \bool_show:n {#1} }

\prgNewFunction \boolVarShow { M } { \bool_show:N #1 }

\prgNewConditional \boolIfExist { M }
  { \bool_if_exist:NTF #1 
    { \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } } }

\prgNewConditional \boolVarIf { M } { \prgReturn {#1} }

\prgNewConditional \boolVarNot { M }
  { \bool_if:NTF #1 
    { \prgReturn { \cFalseBool } } { \prgReturn { \cTrueBool } } }

\prgNewConditional \boolVarAnd { M M }
  { \bool_lazy_and:nnTF {#1} {#2} 
    { \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } } }

\prgNewConditional \boolVarOr { M M }
  { \bool_lazy_or:nnTF {#1} {#2} 
    { \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } } }

\prgNewConditional \boolVarXor { M M }
  { \bool_xor:nnTF {#1} {#2} 
    { \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } } }

\prgNewFunction \boolVarDoUntil { N n }
  { \bool_do_until:Nn #1 {#2} }

\prgNewFunction \boolVarDoWhile { N n }
  { \bool_do_while:Nn #1 {#2} }

\prgNewFunction \boolVarUntilDo { N n }
  { \bool_until_do:Nn #1 {#2} }

\prgNewFunction \boolVarWhileDo { N n }
  {
19.4 Interfaces for Token Lists (Tl)

\bool_while_do:Nn #1 {#2}

\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 \l@FunTmpxTl \tl_new:N \l@FunTmpyTl \tl_new:N \l@FunTmpzTl

\tl_new:N \gTmpaTl \tl_new:N \gTmpbTl \tl_new:N \gTmpcTl
\tl_new:N \g@FunTmpxTl \tl_new:N \g@FunTmpyTl \tl_new:N \g@FunTmpzTl

\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 } { \prgReturn { \expValue #1 } }
\prgNewFunction \tlToStr { m } { \expWhole { \tl_to_str:n { #1 } } }
\prgNewFunction \tlVarToStr { M } { \expWhole { \tl_to_str:N #1 } }
\prgNewFunction \tlConst { M m } { \tl_const:Nn #1 { #2 } }
\prgNewFunction \tlSet { M m } { \tl_gset:Nn #1 {#2} { \tl_set:Nn #1 {#2} } }
\prgNewFunction \tlSetEq { M M } { \tl_gset_eq:NN #1 #2 { \tl_set_eq:NN #1 #2 } }
\prgNewFunction \tlConcat { M M M } { \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} } }

\prgNewFunction \tlPutRight { M m }
{ \_\_fun_do_assignment:Nnn #1
  { \tl_gput_right:Nn #1 (#2) } { \tl_put_right:Nn #1 {#2} } }

\prgNewFunction \tlVarReplaceOnce { M m m }
{ \_\_fun_do_assignment:Nnn #1
  { \tl_greplace_once:Nnn #1 (#2) (#3) } { \tl_replace_once:Nnn #1 {#2} {#3} } }

\prgNewFunction \tlVarReplaceAll { M m m }
{ \_\_fun_do_assignment:Nnn #1
  { \tl_greplace_all:Nnn #1 (#2) (#3) } { \tl_replace_all:Nnn #1 {#2} {#3} } }

\prgNewFunction \tlVarRemoveOnce { M m }
{ \_\_fun_do_assignment:Nnn #1
  { \tl_gremove_once:Nn #1 (#2) } { \tl_remove_once:Nn #1 {#2} } }

\prgNewFunction \tlVarRemoveAll { M m }
{ \_\_fun_do_assignment:Nnn #1
  { \tl_gremove_all:Nn #1 (#2) } { \tl_remove_all:Nn #1 {#2} } }

\prgNewFunction \tlTrimSpaces { m }
{ \expWhole { \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 }
{ \expWhole { \tl_count:n { #1 } } }

\prgNewFunction \tlVarCount { M }
{ \expWhole { \tl_count:N #1 } }

\prgNewFunction \tlHead { m }
{ \expWhole { \tl_head:n { #1 } } }

\prgNewFunction \tlVarHead { M }
{ \expWhole { \tl_head:N #1 } }

\prgNewFunction \tlTail { m }
{ \expWhole { \tl_tail:n { #1 } } }

\prgNewFunction \tlVarTail { M }
{ \expWhole { \tl_tail:N #1 } }

\prgNewFunction \tlItem { m m }
{ \expWhole { \tl_item:nn {#1} {#2} } }

\prgNewFunction \tlVarItem { M m }
{ \expWhole { \tl_item:Nn #1 {#2} } }

\prgNewFunction \tlRandItem { m }
{ \expWhole { \tl_rand_item:n {#1} } }

\prgNewFunction \tlVarRandItem { M }
{ \expWhole { \tl_rand_item:N #1 } }

\prgNewFunction \tlVarCase { M m }
{ \tl_case:Nn {#1} {#2} }

\prgNewFunction \tlVarCaseT { M m n }
{ \tl_case:NnT {#1} {#2} {#3} }

\prgNewFunction \tlVarCaseF { M m n }
{ \tl_case:NnF {#1} {#2} {#3} }

\prgNewFunction \tlVarCaseTF { M m n n }
{ \tl_case:NnTF {#1} {#2} {#3} {#4} }

\prgNewFunction \tlMapInline { m n }
{ \tl_map_inline:nn {#1} {#2} }

\prgNewFunction \tlVarMapInline { M n }
{ \tl_map_inline:Nn {#1} {#2} }

\prgNewFunction \tlMapVariable { m M n }
{ \tl_map_variable:nNn {#1} {#2} {#3} }
\prgNewFunction \tlVarMapVariable { M M n }
   \{ \tl_map_variable:NNn #1 #2 {#3} \}

\prgNewConditional \tlIfExist { M }
   \{ \tl_if_exist:NTF #1
      \{ \prgReturn { \cTrueBool } \} \{ \prgReturn { \cFalseBool } \} \}

\prgNewConditional \tlIfEmpty { m }
   \{ \tl_if_empty:nTF {#1}
      \{ \prgReturn { \cTrueBool } \} \{ \prgReturn { \cFalseBool } \} \}

\prgNewConditional \tlVarIfEmpty { M }
   \{ \tl_if_empty:NTF #1
      \{ \prgReturn { \cTrueBool } \} \{ \prgReturn { \cFalseBool } \} \}

\prgNewConditional \tlIfBlank { m }
   \{ \tl_if_blank:nTF {#1}
      \{ \prgReturn { \cTrueBool } \} \{ \prgReturn { \cFalseBool } \} \}

\prgNewConditional \tlIfEq { m m }
   \{ \tl_if_eq:nnTF {#1} {#2}
      \{ \prgReturn { \cTrueBool } \} \{ \prgReturn { \cFalseBool } \} \}

\prgNewConditional \tlVarIfEq { M M }
   \{ \tl_if_eq:NNTF #1 #2
      \{ \prgReturn { \cTrueBool } \} \{ \prgReturn { \cFalseBool } \} \}

\prgNewConditional \tlIfIn { m m }
   \{ \tl_if_in:nnTF {#1} {#2}
      \{ \prgReturn { \cTrueBool } \} \{ \prgReturn { \cFalseBool } \} \}

\prgNewConditional \tlVarIfIn { M m }
   \{ \tl_if_in:NnTF #1 {#2}
      \{ \prgReturn { \cTrueBool } \} \{ \prgReturn { \cFalseBool } \} \}

\prgNewConditional \tlIfSingle { m }
   \{
\tl_if_single:nTF {#1}
    { \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } }
\}

\prgNewConditional \tlVarIfSingle { M }
{ \tl_if_single:NTF #1
    { \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } }
\}

19.5 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@FunTmpxStr \str_new:N \l@FunTmpyStr \str_new:N \l@FunTmpzStr
\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@FunTmpxStr \str_new:N \g@FunTmpyStr \str_new:N \g@FunTmpzStr

\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 } { \prgReturn { \expValue #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 } { \str_set_eq:NN #1 #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 \strVarReplaceOnce { M m m }
{ \__fun_do_assignment:Nnn #1
  { \str_greplace_once:Nnn #1 {#2} {#3} } { \str_replace_once:Nnn #1 {#2} {#3} }
}

\prgNewFunction \strVarReplaceAll { M m m }
{ \__fun_do_assignment:Nnn #1
  { \str_greplace_all:Nnn #1 {#2} {#3} } { \str_replace_all:Nnn #1 {#2} {#3} }
}

\prgNewFunction \strVarRemoveOnce { M m }
{ \__fun_do_assignment:Nnn #1
  { \str_gremove_once:Nn #1 {#2} } { \str_remove_once:Nn #1 {#2} }
}

\prgNewFunction \strVarRemoveAll { M m }

\__fun_do_assignment:Nnn #1
\{ \str_gremove_all:Nn #1 {#2} \} \{ \str_remove_all:Nn #1 {#2} \}
\}

\prgNewFunction \strCount { m } \{ \expWhole \{ \str_count:n \{ #1 \} \} \}

\prgNewFunction \strVarCount { M } \{ \expWhole \{ \str_count:N \{ #1 \} \} \}

\prgNewFunction \strHead { m } \{ \expWhole \{ \str_head:n \{ #1 \} \} \}

\prgNewFunction \strVarHead { M } \{ \expWhole \{ \str_head:N \{ #1 \} \} \}

\prgNewFunction \strTail { m } \{ \expWhole \{ \str_tail:n \{ #1 \} \} \}

\prgNewFunction \strVarTail { M } \{ \expWhole \{ \str_tail:N \{ #1 \} \} \}

\prgNewFunction \strItem { m m } \{ \expWhole \{ \str_item:nn \{ #1 \} \{ #2 \} \} \}

\prgNewFunction \strVarItem { M m } \{ \expWhole \{ \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 } \{ \str_map_variable:NNn \{ #1 \} \{ #2 \} \{ #3 \} \}

\prgNewConditional \strIfExist { M }\{
\str_if_exist:NTF \{ #1 \}
\{ \prgReturn \{ \cTrueBool \} \} \{ \prgReturn \{ \cFalseBool \} \}
\}\}

\prgNewConditional \strVarIfEmpty { M }\{
\str_if_empty:NTF \{ #1 \}
\{ \prgReturn \{ \cTrueBool \} \} \{ \prgReturn \{ \cFalseBool \} \}
\}\}

\prgNewConditional \strIfEq { m m }\{
\str_if_eq:nnTF \{ #1 \} \{ #2 \}
\{ \prgReturn \{ \cTrueBool \} \} \{ \prgReturn \{ \cFalseBool \} \}
\}\}

\prgNewConditional \strVarIfEq { M M }\{
\}
\begin{verbatim}
\str_if_eq:NNTF #1 #2
{ \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } }
\end{verbatim}

\begin{verbatim}
\prgNewConditional \strIfIn { m m }
{ \str_if_in:nnTF {#1} {#2}
{ \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } }
}
\end{verbatim}

\begin{verbatim}
\prgNewConditional \strVarIfIn { M m }
{ \str_if_in:NnTF #1 {#2}
{ \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } }
}
\end{verbatim}

\begin{verbatim}
\prgNewConditional \strCompare { m N m }
{ \str_compare:nNnTF {#1} #2 {#3}
{ \prgReturn { \cTrueBool } }
{ \prgReturn { \cFalseBool } }
}
\end{verbatim}

19.6 Interfaces for Integers (Int)

\begin{verbatim}
\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 \lTmpiInt \int_new:N \lTmpjInt \int_new:N \lTmpkInt
\int_new:N \l@FunTmpxInt \int_new:N \l@FunTmpyInt \int_new:N \l@FunTmpzInt
\int_new:N \gTmpaInt \int_new:N \gTmpbInt \int_new:N \gTmpcInt
\int_new:N \gTmpiInt \int_new:N \gTmpjInt \int_new:N \gTmpkInt
\int_new:N \g@FunTmpxInt \int_new:N \g@FunTmpyInt \int_new:N \g@FunTmpzInt

\prgNewFunction \intEval { e } { \expWhole { \int_eval:n {#1} } }

\prgNewFunction \intMathAdd { e e } { \expWhole { \int_eval:n { (#1) + (#2) } } }

\prgNewFunction \intMathSub { e e } { \expWhole { \int_eval:n { (#1) - (#2) } } }

\prgNewFunction \intMathMult { e e } { \expWhole { \int_eval:n { (#1) * (#2) } } }

\prgNewFunction \intMathDiv { e e } { \expWhole { \int_div_round:nn {#1} {#2} } }

\prgNewFunction \intMathDivTruncate { e e } { \expWhole { \int_div_truncate:nn {#1} {#2} } }
\end{verbatim}
\prgNewFunction \intMathSign { e } { \expWhole { \int_sign:n {#1} } } \\
\prgNewFunction \intMathAbs { e } { \expWhole { \int_abs:n {#1} } } \\
\prgNewFunction \intMathMax { e e } { \expWhole { \int_max:nn {#1} {#2} } } \\
\prgNewFunction \intMathMin { e e } { \expWhole { \int_min:nn {#1} {#2} } } \\
\prgNewFunction \intMathMod { e e } { \expWhole { \int_mod:nn {#1} {#2} } } \\
\prgNewFunction \intMathRand { e e } { \expWhole { \int_rand:nn {#1} {#2} } } \\
\prgNewFunction \intNew { M } { \int_new:N #1 } \\
\prgNewFunction \intConst { M e } { \int_const:Nn #1 { #2 } } \\
\prgNewFunction \intLog { e } { \int_log:n { #1 } } \\
\prgNewFunction \intVarLog { M } { \int_log:N #1 } \\
\prgNewFunction \intShow { e } { \int_show:n { #1 } } \\
\prgNewFunction \intVarShow { M } { \int_show:N #1 } \\
\prgNewFunction \intUse { M } { \prgReturn { \expValue #1 } } \\
\prgNewFunction \intSet { M e } \\
{ \__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 e }
{ \__fun_do_assignment:Nnn #1 { \int_gadd:Nn #1 {#2} } { \int_add:Nn #1 {#2} } }

\prgNewFunction \intSub { M e }
{ \__fun_do_assignment:Nnn #1 { \int_gsub:Nn #1 {#2} } { \int_sub:Nn #1 {#2} } }

%%%% Command \prg_replicate:nn yields its result after two expansion steps
\prgNewFunction \intReplicate { e m }
{ \exp_args:NNo \exp_args:No \prgReturn \{ \prg_replicate:nn {#1} {#2} \} }

\prgNewFunction \intStepInline { e e e n }
{ \int_step_inline:nnnn {#1} {#2} {#3} {#4} }

\prgNewFunction \intStepOneInline { e e n }
{ \int_step_inline:nnn {#1} {#2} {#3} }

\prgNewFunction \intStepVariable { e e e M n }
{ \int_step_variable:nnnNn {#1} {#2} {#3} #4 {#5} }

\prgNewFunction \intStepOneVariable { e e M n }
{ \int_step_variable:nnNn {#1} {#2} #3 {#4} }

\prgNewConditional \intIfExist { M }
{ \int_if_exist:NTF #1
  \{ \prgReturn \{ \cTrueBool \} \} \{ \prgReturn \{ \cFalseBool \} \} }

\prgNewConditional \intIfOdd { e }
{ \int_if_odd:nTF { #1 }
  \{ \prgReturn \{ \cTrueBool \} \} \{ \prgReturn \{ \cFalseBool \} \} }

\prgNewConditional \intIfEven { e }
{ }
\int_if_even:nTF { #1 } \{ \prgReturn { \cTrueBool } \} \{ \prgReturn { \cFalseBool } \}

\prgNewConditional \intCompare { e \ N \ e } \{
\int_compare:nNnTF {#1} #2 {#3} \{ \prgReturn { \cTrueBool } \} \{ \prgReturn { \cFalseBool } \}
\}

\prgNewFunction \intCase { e \ m } \{ \int_case:nn {#1} {#2} \}
\prgNewFunction \intCaseT { e \ m \ n } \{ \int_case:nnT {#1} {#2} {#3} \}
\prgNewFunction \intCaseF { e \ m \ n } \{ \int_case:nnF {#1} {#2} {#3} \}
\prgNewFunction \intCaseTF { e \ m \ n \ n } \{ \int_case:nnTF {#1} {#2} {#3} {#4} \}

19.7 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 \lTmpiFp \fp_new:N \lTmpjFp \fp_new:N \lTmpkFp
\fp_new:N \l@FunTmpxFp \fp_new:N \l@FunTmpyFp \fp_new:N \l@FunTmpzFp
\fp_new:N \gTmpaFp \fp_new:N \gTmpbFp \fp_new:N \gTmpcFp
\fp_new:N \gTmpiFp \fp_new:N \gTmpjFp \fp_new:N \gTmpkFp
\fp_new:N \g@FunTmpxFp \fp_new:N \g@FunTmpyFp \fp_new:N \g@FunTmpzFp

\prgNewFunction \fpEval { e } \{ \expWhole { \fp_eval:n {#1} } \}
\prgNewFunction \fpMathAdd { e \ e } \{ \expWhole { \fp_eval:n { (#1) + (#2) } } \}
\prgNewFunction \fpMathSub { e \ e } \{ \expWhole { \fp_eval:n { (#1) - (#2) } } \}
\prgNewFunction \fpMathMult { e \ e } \{ \expWhole { \fp_eval:n { (#1) * (#2) } } \}
\prgNewFunction \fpMathDiv { e \ e } \{ \expWhole { \fp_eval:n { (#1) / (#2) } } \}
\prgNewFunction \fpMathSign { e } \{ \expWhole { \fp_sign:n {#1} } \}
\prgNewFunction \fpMathAbs { e } \{ \expWhole { \fp_abs:n {#1} } \}
\prgNewFunction \fpMathMax { e \ e } \{ \expWhole { \fp_max:nn {#1} {#2} } \}
\prgNewFunction \fpMathMin { e \ e } \{ \expWhole { \fp_min:nn {#1} {#2} } \}
\prgNewFunction \fpNew { M } { \fp_new:N #1 }
\prgNewFunction \fpConst { M e } { \fp_const:Nn #1 {#2} }
\prgNewFunction \fpUse { M } { \expWhole { \fp_use:N #1 } }
\prgNewFunction \fpLog { e } { \fp_log:n {#1} }
\prgNewFunction \fpVarLog { M } { \fp_log:N #1 }
\prgNewFunction \fpShow { e } { \fp_show:n {#1} }
\prgNewFunction \fpVarShow { M } { \fp_show:N #1 }
\prgNewFunction \fpSet { M e }
{ \__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 e }
{ \__fun_do_assignment:Nnn #1 { \fp_gadd:Nn #1 {#2} } { \fp_add:Nn #1 {#2} }
}
\prgNewFunction \fpSub { M e }
{ \__fun_do_assignment:Nnn #1 { \fp_gsub:Nn #1 {#2} } { \fp_sub:Nn #1 {#2} }
}
\prgNewFunction \fpStepInline { e e e e n }
{ \fp_step_inline:nnnn {#1} {#2} {#3} {#4} }
\prgNewFunction \fpStepVariable { e e e M n }
{ \fp_step_variable:nnnn {#1} {#2} {#3} #4 {#5} }
19.8 Interfaces for Dimensions (Dim)

\cs_set_eq:NN \cMaxDim \c_{\text{max}}\dim
\cs_set_eq:NN \cZeroDim \c_{\text{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@FunTmpxDim \dim_new:N \l@FunTmpyDim \dim_new:N \l@FunTmpzDim
\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@FunTmpxDim \dim_new:N \g@FunTmpyDim \dim_new:N \g@FunTmpzDim

\prgNewFunction \dimEval { m }
{ \prgReturn { \expWhole \dim_eval:n { #1 } } }

\prgNewFunction \dimMathAdd { m m }
{ \dim_set:Nn \l@FunTmpxDim { \dim_eval:n { (#1) + (#2) } } \prgReturn { \expValue \l@FunTmpxDim } }

\prgNewFunction \dimMathSub { m m }
{ \dim_set:Nn \l@FunTmpxDim { \dim_eval:n { (#1) - (#2) } } \prgReturn { \expValue \l@FunTmpxDim } }

\prgNewFunction \dimMathSign { m }
{ \prgReturn { \expWhole \dim_sign:n { #1 } } }

\prgNewFunction \dimMathAbs { m }
{ \prgReturn { \expWhole \dim_abs:n { #1 } } }

\prgNewFunction \dimMathMax { m m }
{ \prgReturn { \expWhole { \dim_max:nn { #1 } { #2 } } } }

\prgNewFunction \dimMathMin { m m }
{ \prgReturn { \expWhole { \dim_min:nn { #1 } { #2 } } } }

\prgNewFunction \dimMathRatio { m m }
{ \prgReturn { \expWhole { \dim_ratio:nn { #1 } { #2 } } } }

\prgNewFunction \dimNew { M } { \dim_new:N #1 }
\prgNewFunction \dimConst { M m } { \dim_const:Nn #1 {#2} }
\prgNewFunction \dimUse { M } { \prgReturn { \expValue #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 \} \\
\dim_step_inline:nnnn \{ #1 \} \{ #2 \} \{ #3 \} \{ #4 \}

\prgNewFunction \dimStepVariable \{ m m m M n \} \\
\dim_step_variable:nnnNn \{ #1 \} \{ #2 \} \{ #3 \} \{ #4 \} \{ #5 \}

\prgNewConditional \dimIfExist \{ M \} \\
\dim_if_exist:NTF \#1 \\
\prgReturn \{ \cTrueBool \} \{ \prgReturn \{ \cFalseBool \} \}

\prgNewConditional \dimCompare \{ m N m \} \\
\dim_compare:nNnTF \{#1\} \{#2\} \{#3\} \\
\prgReturn \{ \cTrueBool \} \{ \prgReturn \{ \cFalseBool \} \}

\prgNewFunction \dimCase \{ m m \} \\
\dim_case:nn \{#1\} \{#2\} \{#3\} \{#4\}
\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\}

19.9 Interfaces for Sorting Functions (Sort)
\cs_set_eq:NN \sortReturnSame \sort_return_same:
\cs_set_eq:NN \sortReturnSwapped \sort_return_swapped:

19.10 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 \gTmpaClist \clist_new:N \gTmpbClist \clist_new:N \gTmpcClist
\clist_new:N \gTmpiClist \clist_new:N \gTmpjClist \clist_new:N \gTmpkClist
\clist_new:N \l@FunTmpxClist \clist_new:N \g@FunTmpxClist
\clist_new:N \l@FunTmpyClist \clist_new:N \g@FunTmpyClist
\clist_new:N \l@FunTmpzClist \clist_new:N \g@FunTmpzClist
\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 }
\expWhole \clist_use:Nn #1 { #2 }

\prgNewFunction \clistVarJoinExtended { M m m m }
\expWhole \clist_use:Nnnn #1 { #2 } { #3 } { #4 }

\prgNewFunction \clistJoin { m m }
\expWhole \clist_use:nn { #1 } { #2 }

\prgNewFunction \clistJoinExtended { m m m m }
\expWhole \clist_use:nnnn { #1 } { #2 } { #3 } { #4 }

\prgNewFunction \clistConst { M m }
\clist_const:Nn #1 { #2 }

\prgNewFunction \clistSet { M m }
\__fun_do_assignment:Nnn #1
\clist_gset:Nn #1 { #2 }

\prgNewFunction \clistSetEq { M M }
\__fun_do_assignment:Nnn #1
\clist_gset_eq:NN #1 #2

\prgNewFunction \clistSetFromSeq { M M }
\__fun_do_assignment:Nnn #1
\clist_gset_from_seq:NN #1 #2

\prgNewFunction \clistConcat { M M M }
\expWhole \clist_use:nnnn { #1 } { #2 } { #3 } { #4 }
\begin{verbatim}
\newcommand{\clistClear}{\clist_gclear:N \clist_clear:N}
\newcommand{\clistClearNew}{\clist_gclear_new:N \clist_clear_new:N}
\newcommand{\clistPutLeft}{\clist_gput_left:Nn \clist_put_left:Nn}
\newcommand{\clistPutRight}{\clist_gput_right:Nn \clist_put_right:Nn}
\newcommand{\clistVarRemoveDuplicates}{\clist_gremove_duplicates:N \clist_remove_duplicates:N}
\newcommand{\clistVarRemoveAll}{\clist_gremove_all:Nn \clist_remove_all:Nn}
\newcommand{\clistVarReverse}{\clist_greverse:N \clist_reverse:N}
\newcommand{\clistVarSort}{\clist_gsort:Nn \clist_sort:Nn}
\newcommand{\clistCount}{\clist_count:n}
\newcommand{\clistVarCount}{\clist_gcount:Nn #1}
\end{verbatim}
\begin{Verbatim}
{ \expWhole { \clist_count:N \texttt{#1} } }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistGet { \texttt{M M} }
{ \clist_get:NN \texttt{#1} \texttt{#2} }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistGetT { \texttt{M M n} }
{ \clist_get:NNT \texttt{#1} \texttt{#2} \texttt{#3} }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistGetF { \texttt{M M n} }
{ \clist_get:NNF \texttt{#1} \texttt{#2} \texttt{#3} }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistGetTF { \texttt{M M n n} }
{ \clist_get:NNTF \texttt{#1} \texttt{#2} \texttt{#3} \texttt{#4} }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistPop { \texttt{M M} }
{ \__fun_do_assignment:Nnn \texttt{#1} }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistPopT { \texttt{M M n} }
{ \__fun_do_assignment:Nnn \texttt{#1} }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistPopF { \texttt{M M n} }
{ \__fun_do_assignment:Nnn \texttt{#1} }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistPopTF { \texttt{M M n n} }
{ \__fun_do_assignment:Nnn \texttt{#1} }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistPush { \texttt{M m} }
{ \__fun_do_assignment:Nnn \texttt{#1} }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistItem { \texttt{m m} }
{ \expWhole { \clist_item:nn \texttt{#1} \texttt{#2} } }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistVarItem { \texttt{M m} }
{ \expWhole { \clist_item:Nn \texttt{#1} \texttt{#2} } }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistRandItem { \texttt{m} }
{ \expWhole { \clist_rand_item:n \texttt{#1} } }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistVarRandItem { \texttt{M} }
{ \expWhole { \clist_rand_item:N \texttt{#1} } }
\end{Verbatim}

\begin{Verbatim}
\prgNewFunction \clistMapInline { \texttt{m n} }
{ \clist_map_inline:nn \texttt{#1} \texttt{#2} }
\end{Verbatim}
\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}}

\prgNewConditional \clistIfExist { M }
{\clist_if_exist:NTF #1
{\prgReturn {\cTrueBool } } {\prgReturn {\cFalseBool } }}

\prgNewConditional \clistIfEmpty { m }
{\clist_if_empty:nTF {#1}
{\prgReturn {\cTrueBool } } {\prgReturn {\cFalseBool } }}

\prgNewConditional \clistVarIfEmpty { M }
{\clist_if_empty:NTF #1
{\prgReturn {\cTrueBool } } {\prgReturn {\cFalseBool } }}

\prgNewConditional \clistIfIn { m m }
{\clist_if_in:nnTF {#1} {#2}
{\prgReturn {\cTrueBool } } {\prgReturn {\cFalseBool } }}

\prgNewConditional \clistVarIfIn { M m }
{\clist_if_in:NnTF #1 {#2}
{\prgReturn {\cTrueBool } } {\prgReturn {\cFalseBool } }}

19.11 Interfaces for Sequences and Stacks (Seq)
\seq_new:N \TmpaSeq \seq_new:N \TmpbSeq \seq_new:N \TmpcSeq
\seq_new:N \TmpiSeq \seq_new:N \TmpjSeq \seq_new:N \TmpkSeq
\seq_new:N \@FunTmpxSeq \seq_new:N \@FunTmpySeq \seq_new:N \@FunTmpzSeq
\seq_new:N \gTmpaSeq \seq_new:N \gTmpbSeq \seq_new:N \gTmpcSeq
\seq_new:N \gTmpiSeq \seq_new:N \gTmpjSeq \seq_new:N \gTmpkSeq \\
\seq_new:N \g@FunTmpxSeq \seq_new:N \g@FunTmpySeq \seq_new:N \g@FunTmpzSeq \\
\seq_set_eq:NN \cEmptySeq \c_empty_seq \\
\prgNewFunction \seqNew { M } { \seq_new:N #1 } \\
\prgNewFunction \seqVarLog { M } { \seq_log:N #1 } \\
\prgNewFunction \seqVarShow { M } { \seq_show:N #1 } \\
\prgNewFunction \seqVarJoin { M m } \\
{ \expWhole { \seq_use:Nn #1 { #2 } } } \\
\prgNewFunction \seqVarJoinExtended { M m m } \\
{ \expWhole { \seq_use:Nnnn #1 { #2 } { #3 } { #4 } } } \\
\prgNewFunction \seqJoin { m m } \\
{ \expWhole { \seq_use:nn { #1 } { #2 } } } \\
\prgNewFunction \seqJoinExtended { m m m m } \\
{ \expWhole { \seq_use:nnnn { #1 } { #2 } { #3 } { #4 } } } \\
\prgNewFunction \seqConstFromClist { M m } \\
{ \seq_const_from_clist:Nn #1 { #2 } } \\
\prgNewFunction \seqSetFromClist { M m } \\
{ \__fun_do_assignment:Nnn #1 \\
{ \seq_gset_from_clist:Nn #1 {#2} } { \seq_set_from_clist:Nn #1 {#2} } } \\
\prgNewFunction \seqSetEq { M M } \\
{ \__fun_do_assignment:Nnn #1 \\
{ \seq_gset_eq:NN #1 #2 } { \seq_set_eq:NN #1 #2 } } \\
\prgNewFunction \seqSetSplit { M m m } \\
{ \__fun_do_assignment:Nnn #1 \\
{ \seq_gset_split:Nnn #1 {#2} {#3} } { \seq_set_split:Nnn #1 {#2} {#3} } } \\
\prgNewFunction \seqConcat { M M M } \\
{
\texttt{\_\_fun\_do\_assignment:Nnn #1}
\begin{verbatim}
\{ \seq_gconcat:NNN \#1 \#2 \#3 \} \{ \seq_concat:NNN \#1 \#2 \#3 \}
\end{verbatim}

\texttt{\prgNewFunction \seqClear { M }}
\begin{verbatim}
\{ \_\_fun\_do\_assignment:Nnn \#1 \{ \seqgclear:N \#1 \} \{ \seq_clear:N \#1 \}
\end{verbatim}

\texttt{\prgNewFunction \seqClearNew { M }}
\begin{verbatim}
\{ \_\_fun\_do\_assignment:Nnn \#1 \{ \seqgclear\_new:N \#1 \} \{ \seq_clear\_new:N \#1 \}
\end{verbatim}

\texttt{\prgNewFunction \seqPutLeft { M m }}
\begin{verbatim}
\{ \_\_fun\_do\_assignment:Nnn \#1
\{ \seqgput\_left:Nn \#1 \{\#2 \} \} \{ \seq\_put\_left:Nn \#1 \{\#2 \}
\end{verbatim}

\texttt{\prgNewFunction \seqPutRight { M m }}
\begin{verbatim}
\{ \_\_fun\_do\_assignment:Nnn \#1
\{ \seqgput\_right:Nn \#1 \{\#2 \} \} \{ \seq\_put\_right:Nn \#1 \{\#2 \}
\end{verbatim}

\texttt{\prgNewFunction \seqVarRemoveDuplicates { M }}
\begin{verbatim}
\{ \_\_fun\_do\_assignment:Nnn \#1
\{ \seqgremove\_duplicates:N \#1 \} \{ \seq\_remove\_duplicates:N \#1 \}
\end{verbatim}

\texttt{\prgNewFunction \seqVarRemoveAll { M m }}
\begin{verbatim}
\{ \_\_fun\_do\_assignment:Nnn \#1
\{ \seqgremove\_all:Nn \#1 \{\#2 \} \} \{ \seq\_remove\_all:Nn \#1 \{\#2 \}
\end{verbatim}

\texttt{\prgNewFunction \seqVarReverse { M }}
\begin{verbatim}
\{ \_\_fun\_do\_assignment:Nnn \#1 \{ \seqgreverse:N \#1 \} \{ \seq\_reverse:N \#1 \}
\end{verbatim}

\texttt{\prgNewFunction \seqVarSort { M m }}
\begin{verbatim}
\{ \_\_fun\_do\_assignment:Nnn \#1 \{ \seqgsort:Nn \#1 \{\#2 \} \} \{ \seq\_sort:Nn \#1 \{\#2 \}
\end{verbatim}

\texttt{\prgNewFunction \seqVarCount { M }}
\begin{verbatim}
\{ \expWhole { \seq\_count:N \#1 \} \}
\end{verbatim}

\texttt{\prgNewFunction \seqGet { M M }}
\begin{verbatim}
\{ \seq\_get:NN \#1 \#2
\end{verbatim}
\_\_fun\_quark\_upgrade\_no\_value:N \#2

\prgNewFunction \seqGetT \{ M M n \}
{ \seq\_get:NNT \#1 \#2 \{\#3\} }

\prgNewFunction \seqGetF \{ M M n \}
{ \seq\_get:NNF \#1 \#2 \{\#3\} }

\prgNewFunction \seqGetTF \{ M M n n \}
{ \seq\_get:NNTF \#1 \#2 \{\#3\} \{\#4\} }

\prgNewFunction \seqPop \{ M M \}
{
\_\_fun\_do\_assignment:Nnn \#1
\{ \seq\_gpop:NN \#1 \#2 \} \{ \seq\_pop:NN \#1 \#2 \}
\_\_fun\_quark\_upgrade\_no\_value:N \#2
}

\prgNewFunction \seqPopT \{ M M n \}
{
\_\_fun\_do\_assignment:Nnn \#1
\{ \seq\_gpop:NN\_T \#1 \#2 \{\#3\} \} \{ \seq\_pop:NN \#1 \#2 \{\#3\} \}
}

\prgNewFunction \seqPopF \{ M M n \}
{
\_\_fun\_do\_assignment:Nnn \#1
\{ \seq\_gpop:NNF \#1 \#2 \{\#3\} \} \{ \seq\_pop:NNF \#1 \#2 \{\#3\} \}
}

\prgNewFunction \seqPopTF \{ M M n n \}
{
\_\_fun\_do\_assignment:Nnn \#1
\{ \seq\_gpop:NNTF \#1 \#2 \{\#3\} \{\#4\} \} \{ \seq\_pop:NNTF \#1 \#2 \{\#3\} \{\#4\} \}
}

\prgNewFunction \seqPush \{ M m \}
{
\_\_fun\_do\_assignment:Nnn \#1
\{ \seq\_gpush:Nn \#1 \{\#2\} \} \{ \seq\_push:Nn \#1 \{\#2\} \}
}

\prgNewFunction \seqGetLeft \{ M M \}
{
\seq\_get\_left:NN \#1 \#2
\_\_fun\_quark\_upgrade\_no\_value:N \#2
}

\prgNewFunction \seqGetLeftT \{ M M n \}
{ \seq\_get\_left:NNT \#1 \#2 \{\#3\} }

\prgNewFunction \seqGetLeftF \{ M M n \}
{ \seq\_get\_left:NNF \#1 \#2 \{\#3\} }

\prgNewFunction \seqGetLeftTF \{ M M n n \}
{ \seq\_get\_left:NNTF \#1 \#2 \{\#3\} \{\#4\} }

\prgNewFunction \seqGetRight \{ M M \}
{
\seq\_get\_right:NN \#1 \#2
\_\_fun\_quark\_upgrade\_no\_value:N \#2
}

\prgNewFunction \seqGetRightT \{ M M n \}
{ \seq\_get\_right:NNT \#1 \#2 \{\#3\} }

\prgNewFunction \seqGetRightF \{ M M n \}
\prgNewFunction \seqGetRightTF { M M n n } { \seq_get_right:NNTF #1 #2 {#3} {#4} }
\prgNewFunction \seqPopLeft { M M } { \_\fun_do_assignment:Nnn #1 { \seq_gpop_left:NN #1 #2 } { \seq_pop_left:NN #1 #2 } \_\fun_quark_upgrade_no_value:N #2 }
\prgNewFunction \seqPopLeftT { M M n } { \_\fun_do_assignment:Nnn #1 { \seq_gpop_left:NNT #1 #2 {#3} } { \seq_pop_left:NNT #1 #2 {#3} } }
\prgNewFunction \seqPopLeftF { M M n } { \_\fun_do_assignment:Nnn #1 { \seq_gpop_left:NNF #1 #2 {#3} } { \seq_pop_left:NNF #1 #2 {#3} } }
\prgNewFunction \seqPopLeftTF { M M n n } { \_\fun_do_assignment:Nnn #1 { \seq_gpop_left:NNTF #1 #2 {#3} {#4} } { \seq_pop_left:NNTF #1 #2 {#3} {#4} } }
\prgNewFunction \seqPopRight { M M } { \_\fun_do_assignment:Nnn #1 { \seq_gpop_right:NN #1 #2 } { \seq_pop_right:NN #1 #2 } \_\fun_quark_upgrade_no_value:N #2 }
\prgNewFunction \seqPopRightT { M M n } { \_\fun_do_assignment:Nnn #1 { \seq_gpop_right:NNT #1 #2 {#3} } { \seq_pop_right:NNT #1 #2 {#3} } }
\prgNewFunction \seqPopRightF { M M n } { \_\fun_do_assignment:Nnn #1 { \seq_gpop_right:NNF #1 #2 {#3} } { \seq_pop_right:NNF #1 #2 {#3} } }
\prgNewFunction \seqPopRightTF { M M n n } { \_\fun_do_assignment:Nnn #1 { \seq_gpop_right:NNTF #1 #2 {#3} {#4} } { \seq_pop_right:NNTF #1 #2 {#3} {#4} } }
\prgNewFunction \seqVarItem { M m } { \expWhole { \seq_item:Nn #1 {#2} } }
\prgNewFunction \seqVarRandItem { M } { \expWhole { \seq_rand_item:N #1 } }
\prgNewFunction \seqVarMapInline { M n }
19.12 Interfaces for Property Lists (Prop)

\prop_new:N \lTmpaProp \prop_new:N \lTmpbProp \prop_new:N \lTmpcProp
\prop_new:N \lTmpiProp \prop_new:N \lTmpjProp \prop_new:N \lTmpkProp
\prop_new:N \l@FunTmpxProp \prop_new:N \l@FunTmpyProp \prop_new:N \l@FunTmpzProp
\prop_new:N \gTmpaProp \prop_new:N \gTmpbProp \prop_new:N \gTmpcProp
\prop_new:N \gTmpiProp \prop_new:N \gTmpjProp \prop_new:N \gTmpkProp
\prop_new:N \g@FunTmpxProp \prop_new:N \g@FunTmpyProp \prop_new:N \g@FunTmpzProp
\prop_const_from_keyval:Nn \cEmptyProp {}
\prgNewFunction \propSetEq { M M }
{ \__fun_do_assignment:Nnn #1
  { \prop_gset_eq:NN #1 #2 } { \prop_set_eq:NN #1 #2 } }

\prgNewFunction \propClear { M }
{ \__fun_do_assignment:Nnn #1
  { \prop_gclear:N #1 } { \prop_clear:N #1 } }

\prgNewFunction \propClearNew { M }
{ \__fun_do_assignment:Nnn #1
  { \prop_gclear_new:N #1 } { \prop_clear_new:N #1 } }

\prgNewFunction \propConcat { M M M }
{ \__fun_do_assignment:Nnn #1
  { \prop_gconcat:NNN #1 #2 #3 } { \prop_concat:NNN #1 #2 #3 } }

\prgNewFunction \propPut { M m m }
{ \__fun_do_assignment:Nnn #1
  { \prop_gput:Nnn #1 {#2} {#3} } { \prop_put:Nnn #1 {#2} {#3} } }

\prgNewFunction \propPutIfNew { M m m }
{ \__fun_do_assignment:Nnn #1
  { \prop_gput_if_new:Nnn #1 {#2} {#3} } { \prop_put_if_new:Nnn #1 {#2} {#3} } }

\prgNewFunction \propPutFromKeyval { M m }
{ \__fun_do_assignment:Nnn #1
  { \prop_gput_from_keyval:Nn #1 {#2} } { \prop_put_from_keyval:Nn #1 {#2} } }

\prgNewFunction \propVarRemove { M m }
{ \__fun_do_assignment:Nnn #1
  { \prop_gremove:Nn #1 {#2} } { \prop_remove:Nn #1 {#2} } }

\prgNewFunction \propVarCount { M } \expWhole { \prop_count:N #1 }

\prgNewFunction \propVarItem { M m } \expWhole { \prop_item:Nn #1 {#2} }

\prgNewFunction \propToKeyval { M } \expWhole { \prop_to_keyval:N #1 }
\prgNewFunction \propGet { M m M } {
 \prop_get:NnN #1 {#2} #3
 \__fun_quark_upgrade_no_value:N #3
}
\prgNewFunction \propGetT { M m M n } { \prop_get:NnNT #1 {#2} #3 {#4} }
\prgNewFunction \propGetF { M m M n } { \prop_get:NnNF #1 {#2} #3 {#4} }
\prgNewFunction \propGetTF { M m M n n } { \prop_get:NnNTF #1 {#2} #3 {#4} {#5} }

\prgNewFunction \propPop { M m M } {
 \__fun_do_assignment:Nnn #1
 { \prop_gpop:NnN #1 {#2} #3 }
 \__fun_quark_upgrade_no_value:N #3
}
\prgNewFunction \propPopT { M m M n } {
 \__fun_do_assignment:Nnn #1
 { \prop_gpop:NnNT #1 {#2} #3 {#4} }
}
\prgNewFunction \propPopF { M m M n } {
 \__fun_do_assignment:Nnn #1
 { \prop_gpop:NnNF #1 {#2} #3 {#4} }
}
\prgNewFunction \propPopTF { M m M n n } {
 \__fun_do_assignment:Nnn #1
 { \prop_gpop:NnNTF #1 {#2} #3 {#4} {#5} }
}
\prgNewFunction \propVarMapInline { M n } { \prop_map_inline:Nn #1 {#2} }
\cs_set_eq:NN \propMapBreak \prop_map_break:
\prgNewConditional \propIfExist { M } {
 \prop_if_exist:NTF #1
 { \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } }
}
\prgNewConditional \propVarIfEmpty { M } {
 \prop_if_empty:NTF #1
 { \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } }
}
\prgNewConditional \propVarIfIn { M m } {
 \prop_if_in:NnTF #1 {#2}
 { \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } }
}
19.13 Interfaces for Regular Expressions (Regex)

\regex_new:N \lTmpaRegex  \regex_new:N \lTmpbRegex  \regex_new:N \lTmpcRegex
\regex_new:N \lT mpiRegex  \regex_new:N \lTmpjRegex  \regex_new:N \lTmpkRegex
\regex_new:N \gTmpaRegex  \regex_new:N \gTmpbRegex  \regex_new:N \gTmpcRegex
\regex_new:N \gTmpiRegex  \regex_new:N \gTmpjRegex  \regex_new:N \gTmpkRegex
\regex_new:N \l@FunTmpxRegex  \regex_new:N \g@FunTmpxRegex
\regex_new:N \l@FunTmpyRegex  \regex_new:N \g@FunTmpyRegex
\regex_new:N \l@FunTmpzRegex  \regex_new:N \g@FunTmpzRegex

\prgNewFunction \regexNew { M } { \regex_new:N #1 }

\prgNewFunction \regexSet { M m } 
{ \_fun_do_assignment:Nnn #1 
  { \regex_gset:Nn #1 {#2} } { \regex_set:Nn #1 {#2} } 
}

\prgNewFunction \regexConst { M m } { \regex_const:Nn #1 {#2} }
\prgNewFunction \regexLog { m } { \regex_log:n {#1} }
\prgNewFunction \regexVarLog { M } { \regex_log:N #1 }
\prgNewFunction \regexShow { m } { \regex_show:n {#1} }
\prgNewFunction \regexVarShow { M } { \regex_show:N #1 }

\prgNewConditional \regexMatch { m m } 
{ \regex_match:nnTF {#1} {#2} 
  { \prgReturn { \cTrueBool } }  { \prgReturn { \cFalseBool } } 
}

\prgNewConditional \regexVarMatch { M m } 
{ \regex_match:NnTF #1 {#2} 
  { \prgReturn { \cTrueBool } }  { \prgReturn { \cFalseBool } } 
}

\prgNewFunction \regexCount { m m M } { \regex_count:nnN {#1} {#2} #3 }
\prgNewFunction \regexVarCount { M m M } { \regex_count:NnN #1 {#2} #3 }

\prgNewFunction \regexMatchCase { m m } 
{ \regex_match_case:nn {#1} {#2} }
\prgNewFunction \regexMatchCaseT { m m n } 
{ \regex_match_case:nnT {#1} {#2} {#3} }
\prgNewFunction \regexMatchCaseF { m m n } 
{ \regex_match_case:nnF {#1} {#2} {#3} }
\prgNewFunction \regexMatchCaseTF { m m n n } 
{ \regex_match_case:nnTF {#1} {#2} {#3} {#4} }
\prgNewFunction \regexExtractOnce { m m M } 
{ \regex_extract_once:nnN {#1} {#2} #3 }
\prgNewFunction \regexExtractOnceT { m m M n } 
{ \regex_extract_once:nnNT {#1} {#2} #3 {#4} }
\prgNewFunction \regexExtractOnceF { m m M n } 
{ \regex_extract_once:nnNF {#1} {#2} #3 {#4} }
\prgNewFunction \regexExtractOnceTF { m m M n n } 
{ \regex_extract_once:nnNTF {#1} {#2} #3 {#4} {#5} }
\prgNewFunction \regexVarExtractOnce { M m M } 
{ \regex_extract_once:NnN #1 {#2} #3 }
\prgNewFunction \regexVarExtractOnceT { M m M n } 
{ \regex_extract_once:NnNT #1 {#2} #3 {#4} }
\prgNewFunction \regexVarExtractOnceF { M m M n } 
{ \regex_extract_once:NnNF #1 {#2} #3 {#4} }
\prgNewFunction \regexVarExtractOnceTF { M m M n n } 
{ \regex_extract_once:NnNTF #1 {#2} #3 {#4} {#5} }
\prgNewFunction \regexExtractAll { m m M } 
{ \regex_extract_all:nnN {#1} {#2} #3 }
\prgNewFunction \regexExtractAllT { m m M n } 
{ \regex_extract_all:nnNT {#1} {#2} #3 {#4} }
\prgNewFunction \regexExtractAllF { m m M n } 
{ \regex_extract_all:nnNF {#1} {#2} #3 {#4} }
\prgNewFunction \regexExtractAllTF { m m M n n } 
{
\regex_extract_all:nnTF {#1} {#2} #3 {#4} {#5}
\
\prgNewFunction \regexVarExtractAll { M m M }
{ \regex_extract_all:NnN #1 {#2} #3 }
\prgNewFunction \regexVarExtractAllT { M m M n }
{ \regex_extract_all:NnNT #1 {#2} #3 {#4} }
\prgNewFunction \regexVarExtractAllF { M m M n }
{ \regex_extract_all:NnNF #1 {#2} #3 {#4} }
\prgNewFunction \regexVarExtractAllTF { M m M n n }
{ \regex_extract_all:NnNTF #1 {#2} #3 {#4} {#5} }
\
\prgNewFunction \regexSplit { m m M }
{ \regex_split:nnN {#1} {#2} #3 }
\prgNewFunction \regexSplitT { m m M n }
{ \regex_split:nnNT {#1} {#2} #3 {#4} }
\prgNewFunction \regexSplitF { m m M n }
{ \regex_split:nnNF {#1} {#2} #3 {#4} }
\prgNewFunction \regexSplitTF { m m M n n }
{ \regex_split:nnNTF {#1} {#2} #3 {#4} {#5} }
\
\prgNewFunction \regexVarSplit { M m M }
{ \regex_split:NnN #1 {#2} #3 }
\prgNewFunction \regexVarSplitT { M m M n }
{ \regex_split:NnNT #1 {#2} #3 {#4} }
\prgNewFunction \regexVarSplitF { M m M n }
{ \regex_split:NnNF #1 {#2} #3 {#4} }
\prgNewFunction \regexVarSplitTF { M m M n n }
{ \regex_split:NnNTF #1 {#2} #3 {#4} {#5} }
\
\prgNewFunction \regexReplaceOnce { m m M }
{ \regex_replace_once:nnN {#1} {#2} #3 }
\prgNewFunction \regexReplaceOnceT { m m M n }
{
  \regex_replace_once:nnNT {#1} {#2} #3 {#4}
}
\prgNewFunction \regexReplaceOnceF { m m M n }
{
  \regex_replace_once:nnNF {#1} {#2} #3 {#4}
}
\prgNewFunction \regexReplaceOnceTF { m m M n n }
{
  \regex_replace_once:nnNTF {#1} {#2} #3 {#4} {#5}
}
\prgNewFunction \regexVarReplaceOnce { M m M }
{
  \regex_replace_once:NnN #1 {#2} #3
}
\prgNewFunction \regexVarReplaceOnceT { M m M n }
{
  \regex_replace_once:NnNT #1 {#2} #3 {#4}
}
\prgNewFunction \regexVarReplaceOnceF { M m M n }
{
  \regex_replace_once:NnNF #1 {#2} #3 {#4}
}
\prgNewFunction \regexVarReplaceOnceTF { M m M n n }
{
  \regex_replace_once:NnNTF #1 {#2} #3 {#4} {#5}
}
\prgNewFunction \regexReplaceAll { m m M }
{
  \regex_replace_all:nnN {#1} {#2} #3
}
\prgNewFunction \regexReplaceAllT { m m M n }
{
  \regex_replace_all:nnNT {#1} {#2} #3 {#4}
}
\prgNewFunction \regexReplaceAllF { m m M n }
{
  \regex_replace_all:nnNF {#1} {#2} #3 {#4}
}
\prgNewFunction \regexReplaceAllTF { m m M n n }
{
  \regex_replace_all:nnNTF {#1} {#2} #3 {#4} {#5}
}
\prgNewFunction \regexVarReplaceAll { M m M }
{
  \regex_replace_all:NnN #1 {#2} #3
}
\prgNewFunction \regexVarReplaceAllT { M m M n }
{
  \regex_replace_all:NnNT #1 {#2} #3 {#4}
}
\prgNewFunction \regexVarReplaceAllF { M m M n }

\begin{verbatim}
{ \regex_replace_all:NnNF #1 {#2} #3 {#4}

\prgNewFunction \regexVarReplaceAllTF { M m n n }
{ \regex_replace_all:NnTF #1 {#2} #3 {#4} {#5}

\prgNewFunction \regexReplaceCaseOnce { m M }
{ \regex_replace_case_once:nN {#1} #2

\prgNewFunction \regexReplaceCaseOnceT { m M n }
{ \regex_replace_case_once:nN {#1} #2 {#3}

\prgNewFunction \regexReplaceCaseOnceF { m M n }
{ \regex_replace_case_once:nN {#1} #2 {#3}

\prgNewFunction \regexReplaceCaseOnceTF { m M n n }
{ \regex_replace_case_once:nN {#1} #2 {#3} {#4}

\prgNewFunction \regexReplaceCaseAll { m M }
{ \regex_replace_case_all:nN {#1} #2

\prgNewFunction \regexReplaceCaseAllT { m M n }
{ \regex_replace_case_all:nN {#1} #2 {#3}

\prgNewFunction \regexReplaceCaseAllF { m M n }
{ \regex_replace_case_all:nN {#1} #2 {#3}

\prgNewFunction \regexReplaceCaseAllTF { m M n n }
{ \regex_replace_case_all:nN {#1} #2 {#3} {#4}

\end{verbatim}

19.14 Interfaces for Token Manipulation (Token)

\begin{verbatim}
\prgNewFunction \charLowercase { M } { \expWhole { \char_lowercase:N #1 } }
\prgNewFunction \charUppercase { M } { \expWhole { \char_uppercase:N #1 } }
\prgNewFunction \charTitlecase { M } { \expWhole { \char_titlecase:N #1 } }
\prgNewFunction \charFoldcase { M } { \expWhole { \char_foldcase:N #1 } }
\prgNewFunction \charStrLowercase { M } { \expWhole { \char_str_lowercase:N #1 } }
\prgNewFunction \charStrUppercase { M } { \expWhole { \char_str_uppercase:N #1 } }
\end{verbatim}
19.15 Interfaces for Text Processing (Text)

\prgNewFunction \textExpand { m } 
{ 
  \expWhole { \text\_expand:n {#1} } 
}

\prgNewFunction \textLowercase { m } 
{ 
  \expWhole { \text\_lowercase:n {#1} } 
}

\prgNewFunction \textUppercase { m } 
{ 
  \expWhole { \text\_uppercase:n {#1} } 
}

\prgNewFunction \textTitlecase { m } 
{ 
  \expWhole { \text\_titlecase:n {#1} } 
}

\prgNewFunction \textTitlecaseFirst { m } 
{ 
  \expWhole { \text\_titlecase\_first:n {#1} } 
}

\prgNewFunction \textLangLowercase { m m } 
{ 
  \expWhole { \text\_lang\_lowercase:nn {#1} {#2} } 
}

\prgNewFunction \textLangUppercase { m m } 
{ 
  \expWhole { \text\_lang\_uppercase:nn {#1} {#2} } 
}

\prgNewFunction \textLangTitlecase { m m } 
{ 
  \expWhole { \text\_lang\_titlecase:nn {#1} {#2} } 
}
19.16 Interfaces for Files (File)

\msg_new:nnn { functional } { file-not-found } { File "#1" not found! }

\prgNewFunction \fileInput { m }
{ \file_get:nnN {#1} {} \l@FunTmpxTl \quark_if_no_value:NTF \l@FunTmpxTl { \msg_error:nnn \l@FunTmpxTl \l@FunTmpxTl } \tlUse \l@FunTmpxTl }

\prgNewFunction \fileIfExistInput { m }
{ \file_get:nnN {#1} {} \l@FunTmpxTl \quark_if_no_value:NF \l@FunTmpxTl \tlUse \l@FunTmpxTl }

\prgNewFunction \fileIfExistInputF { m n }
{ \file_get:nnN {#1} {} \l@FunTmpxTl \quark_if_no_value:NTF \l@FunTmpxTl \tlUse \l@FunTmpxTl \l@FunTmpxTl }

\cs_set_eq:NN \fileInputStop \file_input_stop:

\prgNewFunction \fileGet { m m M }
{ \file_get:nnN {#1} {#2} \l@FunTmpxTl \__fun_quark_upgrade_no_value:N \l@FunTmpxTl #3 }

\prgNewFunction \fileGetT { m m n }
{ \file_get:nnNT {#1} {#2} #3 {#4} }

\prgNewFunction \fileGetF { m m n }
{ \file_get:nnNF {#1} {#2} #3 {#4} }

\prgNewFunction \fileGetTF { m m n n }
{ \file_get:nnNTF {#1} {#2} #3 {#4} {#5} }

\prgNewConditional \fileIfExist { m }
{...}
19.17 Interfaces for Quarks (Quark)

\quark_new:N \qNoValue

\cs_new_protected:Npn \__fun_quark_upgrade_no_value:N #1
  { \quark_if_no_value:NT #1 { \tl_set_eq:NN #1 \qNoValue } }

\prgNewConditional \quarkVarIfNoValue { M }
  { \tl_if_eq:NNTF \qNoValue #1
    { \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } } }

19.18 Interfaces to Legacy Concepts (Legacy)

\prgNewConditional \legacyIf { m }
  { \legacy_if:nTF {#1}
    { \prgReturn { \cTrueBool } } { \prgReturn { \cFalseBool } } }

\prgNewFunction \legacyIfSetTrue { m }
  { \__fun_do_assignment:Nnn \c@name
    { \legacy_if_gset_true:n {#1} } { \legacy_if_set_true:n {#1} } }

\prgNewFunction \legacyIfSetFalse { m }
  { \__fun_do_assignment:Nnn \c@name
    { \legacy_if_gset_false:n {#1} } { \legacy_if_set_false:n {#1} } }

\prgNewFunction \legacyIfSet { m m }
  { \__fun_do_assignment:Nnn \c@name
    { \legacy_if_gset:nn {#1} {#2} } { \legacy_if_set:nn {#1} {#2} } }