String Processing in Maxima 5.9.1

Volker van Nek, van.Nek@gmx.net, Aachen, 30.10.2005

'stringproc1.lisp' enlarges Maximas capabilities of working with strings. If you find bugs or if you need more functions, feel free to mail. If you use it for education, please let me know.

For installation just copy 'stringproc1.lisp' to a folder in path `file_search_lisp`, possibly '…/Maxima-5.9.1/share/maxima/5.9.1/share'. Load 'stringproc1.lisp' by typing `load("stringproc1");`.

Remark: 'stringproc1.lisp' is written for Maxima 5.9.1. Version 5.9.2 should use 'stringproc2.lisp'.

Introduction

In Maxima a string is easily constructed by typing `"text"`. Note that Maxima-strings are no Lisp-strings and vice versa. Tests can be done with `stringp` respectively `lstringp`. If for some reasons you have a value, that is a Lisp-string, maybe when using Maxima-function `sconcat`, you can convert via `sunlisp`.

```
(C1) m: "text";
(D1)                                    text
(C2) [stringp(m),lstringp(m)];
(D2)                                [TRUE, FALSE]
(C3) l: sconcat("text");
(D3)                                    text
(C4) [stringp(l),lstringp(l)];
(D4)                                [FALSE, TRUE]
(C5) stringp( sunlisp(l) );
(D5)                                    TRUE
```

All functions in 'stringproc1.lisp', that return strings, return Maxima-strings.

Characters are introduced as Maxima-strings of length 1. Of course, these are no Lisp-characters. Tests can be done with `charp` (respectively `lcharp` and conversion from Lisp to Maxima with `cunlisp`).

```
(C1) c: "e";
(D1)                                      e
(C2) [charp(c),lcharp(c)];
(D2)                                [TRUE, FALSE]
(C3) supcase(c);
(D3)                                      E
(C4) charp(%);
(D4)                                    TRUE
```

Again, all functions in 'stringproc1.lisp', that return characters, return Maxima-characters. Due to the fact, that the introduced characters are strings of length 1, you can use a lot of string functions also for characters. As seen, `supcase` is one example.

It is important to know, that the first character in a Maxima-string is at position 1. This is designed due to the fact that the first element in a Maxima-list is at position 1 too. See definitions of `charat` and `charlist` for examples.

In applications string-functions are often used when working with files. You will find some useful stream- and print-functions in 'stringproc1.lisp'. The following example shows some of the here introduced functions at work. They are marked with green color.

Example:

Let `file` contain Maxima console I/O, saved with 'Save Console to File' or with copy and paste. `extracti` then extracts the values of all input labels to a `batch`able file, which path is the return value. The batch process can directly be started with `batch(%)`.
Note that `extracti` fails if at least one label is damaged, maybe due to erasing the ')'. It fails too, if there are input lines from a batch process. In this case terminators are missing.

```
extracti(file):= block(
   [ s1: openr(file), ifile: sconc(file,".in"), line, nl: false ],
   s2: openw(ifile),

   while ( stringp(line: readline(s1)) ) do (
      if ssearch( sconc("(",inchar),line ) = 1 then (
         line: strim(" ",substring( line,ssearch(")",line)+1 )),
         printf( s2,"~a~%",line ),
         checklast(line) )
      else if nl then (
         line: strimr(" ",line),
         printf( s2,"~a~%",line ),
         checklast(line) )),

   close(s1), close(s2),
   ifile)$

checklast(line):= block(
   [ last: charat( line,slength(line) ) ],
   if cequal(last,";") or cequal(last,"$") then
      nl:false else nl:true )$
```

File 'C:\home\maxima\test.out':

```
(C1) f(x):= sin(x)$
(C2) diff(f(x),x);
(D2)                                  cos(x)
(C3) df(x):= ''%;
(D3)                              df(x) := cos(x)
(C4) df(0);
(D4)                                     1
```

Maxima:

```
(C11) extracti("C:\\home\\maxima\\test.out");
(D11)                C:\home\maxima\test.out.in
(C12) batch(%);

batching #pC:/home/maxima/test.out.in
(C13)                              f(x) := SIN(x)
(C14)                               DIFF(f(x), x)
(D14)                                  COS(x)
(C15)                              df(x) := COS(x)
(D15)                              df(x) := COS(x)
(C16)                                   df(0)
(D16)                                     1
```

Definitions for Input and Output:

Example:

```
(C1) s: openw("C:\\home\\file.txt");
(D1)                    #<output stream C:\home\file.txt>
(C2) control: "~2tAn atom: ~20t~a~%~2tand a list: ~20t~{~r ~}~%~2tand an
integer: ~20t~d~%"$
(C3) printf( s,control, 'true,[1,2,3],42 )$
(D3)                                    FALSE
(C4) close(s);
(D4)                                    TRUE
(C5) s: openr("C:\\home\\file.txt");
(D5)                    #<input stream C:\home\file.txt>
(C6) while stringp( tmp:readline(s) ) do print(tmp)$
  An atom:           TRUE
  and a list:        one two three
  and an integer:    42
(C7) close(s)$
```

Function: close (stream)
Closes stream and returns true if stream had been open.

Function: flength (stream)
Returns the number of elements in stream.

Function: fposition (stream)
Function: fposition (stream,pos)
Returns the current position in stream, if pos is not used. If pos is used, fposition sets the position in stream. pos has to be a positive number, the first element in stream is in position 1.

Function: freshline ()
Function: freshline (stream)
Writes a new line to stream, if the position is not at the beginning of a line. If stream is not used, false is the default.

Function: newline ()
Function: newline (stream)
Writes a new line to stream, if the position is not at the beginning of a line. If stream is not used, false is the default.

Function: opena (file)
Returns an output stream to file. If an existing file is opened, opena appends elements at the end of file.

Function: openr (file)
Returns an input stream to file. If file does not exist, it will be created.

Function: openw (file)
Returns an output stream to file. If file does not exist, it will be created. If an existing file is opened, openw destructively modifies file.

Function: printf (dest,string)
Function: printf (dest,string,expr1,expr2,..)
printf is like format in Common Lisp.
(gcl.info: format produces formatted output by outputting the characters of control-string string and observing that a tilde introduces a directive. The character after the tilde, possibly preceded by prefix parameters and modifiers, specifies what kind of formatting is desired. Most directives use one or more elements of args to create their output.)

The following description and the examples may give an idea of using printf. See Lisp reference for more information. Note that there are some directives, which do not work in Maxima. For example, ~:[fails. printf is designed with the intention, that ~s is read as ~a. Also note that the selection directive ~[is zero-indexed.

```
~%      new line
~&      fresh line
~t      tab
~$      monetary
~d      decimal integer
~b      binary integer
~o      octal integer
~x      hexadecimal integer
~br     base-b integer
~r      spell an integer
~p      plural
~f      floating point
~e      scientific notation
~g      ~f or ~e, depending upon magnitude
~a      as printed by Maxima function print
~s      like ~a
~~      ~
~<      justification, ~> terminates
~(      case conversion, ~) terminates
~[      selection, ~] terminates
~{      iteration, ~} terminates
```

```
(C1) printf( false, "~s ~a ~4f ~a ~@r",
"String",sym,bound,sqrt(8),144), bound = 1.234;
(D1)                    String sym 1.23 2*SQRT(2) CXLIV
(C2) printf( false,"~{~a ~}",["one",2,"THREE"] );
(D2)                               one 2 THREE
(C3) printf( true,"~{~{~9,1f ~}~%~}",mat ),
mat = args( matrix([1.1,2,3.33],[4,5,6],[7,8.88,9]) )$
      1.1       2.0       3.3
      4.0       5.0       6.0
      7.0       8.9       9.0
(C4) control: "~:(~r~) bird~p ~[is~;are~] singing."$
(C5) printf( false,control, n,n,if n=1 then 0 else 1 ), n=2;
(D5)                   Two birds are singing.
```

If `dest` is a stream or `true`, then `printf` returns `false`. Otherwise, `printf` returns a string containing the output.

`Function: readline (stream)`
Returns a string containing the characters from the current position in `stream` up to the end of the line or `false` if the end of the file is encountered.

Definitions for Characters:

`Function: alphacharp (char)`
Returns `true` if `char` is an alphabetic character.

`Function: alphanumericp (char)`
Returns `true` if `char` is an alphabetic character or a digit.

`Function: ascii (int)`
Returns the character corresponding to the ASCII number `int`. (-1 < `int` < 256)

```
(C1) for n from 0 thru 255 do ( tmp: ascii(n),
if alphacharp(tmp) then sprint(tmp) )$
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k
l m n o p q r s t u v w x y z
```

`Function: cequal (char1,char2)`
Returns `true` if `char1` and `char2` are the same.

`Function: cequalignore (char1,char2)`
Like `cequal` but ignores case.

`Function: cgreaterp (char1,char2)`
Returns `true` if the ASCII number of `char1` is greater than the number of `char2`.

`Function: cgreaterpignore (char1,char2)`
Like `cgreaterp` but ignores case.

`Function: charp (obj)`
Returns `true` if `obj` is a Maxima-character. See introduction for example.

`Function: cint (char)`
Returns the ASCII number of `char`.

`Function: clessp (char1,char2)`
Returns `true` if the ASCII number of `char1` is less than the number of `char2`.

`Function: clesspignore (char1,char2)`
Like `clessp` but ignores case.

Function: constituent (char)
Returns `true` if `char` is a graphic character and not the space character. A graphic character is a character one can see, plus the space character.
(`constituent` is defined by Paul Graham, ANSI Common Lisp, 1996, page 67.)

```
(C1) for n from 0 thru 255 do ( tmp: ascii(n),
if constituent(tmp) then sprint(tmp) )$
! " # % ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F
G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _ ` a b c d e f g h i j k
l m n o p q r s t u v w x y z { | } ~
```

Function: cunlisp (lisp-char)
Converts a Lisp-character into a Maxima-character. (You won't need it.)

Function: digitcharp (char)
Returns `true` if `char` is a digit.

Function: lcharp (obj)
Returns `true` if `obj` is a Lisp-character. (You won't need it.)

Function: lowercasep (char)
Returns `true` if `char` is a lowercase character.

Variable: newline
The character 'newline'.

Variable: space
The character 'space'.

Variable: tab
The character 'tab'.

Function: uppercasep (char)
Returns `true` if `char` is an uppercase character.

Definitions for Strings:

Function: sunlisp (lisp-string)
Converts a Lisp-string into a Maxima-string. (In general you won't need it.)

Function: lstringp (obj)
Returns `true` if `obj` is a Lisp-string. (In general you won't need it.)

Function: stringp (obj)
Returns `true` if `obj` is a Maxima-string. See introduction for example.

Function: charat (string,n)
Returns the `n`th character of `string`. The first character in `string` is returned with n = 1.

```
(C1) charat("Lisp",1);
(D1)                                 L
```

Function: charlist (string)
Returns the list of all characters in `string`.

```
(C2) charlist("Lisp");
(D2)                            [L, i, s, p]
(C3) %[1];
(D3)                                 L
```

Function: eval_string (string)

Function by Robert Dodier, Maxima 5.9.2, eval_string.lisp, Oct 2005. Modified for 5.9.1.

`eval_string` parses the Maxima string `string` as a Maxima expression and evaluate it. `string` is a Maxima string. It may or may not have a terminator (dollar sign `$' or semicolon `;'). Only the first expression is parsed and evaluated, if there is more than one.

```
(C1) eval_string("foo: 42; bar: foo^2 + baz");
(D1)                                  42
(C2) eval_string("(foo: 42, bar: foo^2 + baz)");
(D2)                               baz + 1764
```

Function: parsetoken (string)

`parsetoken` converts the first token in `string` to the corresponding number or returns `false` if the number cannot be determined . The delimiter set for tokenizing is {space, comma, semicolon, tab, newline}.

```
(C1) 2*parsetoken("1.234 5.678");
(D1)                                  2.468
```

Function: parse_string (string)

Function by Robert Dodier, Maxima 5.9.2, eval_string.lisp, Oct 2005. Modified for 5.9.1.

`parse_string` parses the Maxima string `string` as a Maxima expression (do not evaluate it). `string` is a Maxima string. It may or may not have a terminator (dollar sign `$' or semicolon `;'). Only the first expression is parsed, if there is more than one.

```
(C1) parse_string("foo: 42; bar: foo^2 + baz");
(D1)                                foo : 42
(C2) parse_string("(foo: 42, bar: foo^2 + baz)");
                                                  2
(D2)                      (foo : 42, bar : foo  + baz)
```

Function: sconc (arg1,arg2,...)

Evaluates its arguments and concatenates them into a string. `sconc` is like `sconcat` but returns a Maxima string.

```
(C1) sconc("xx[",3,"]:",expand((x+y)^3));
(D1)                    xx[3]:y^3+3*x*y^2+3*x^2*y+x^3
(C2) stringp(%);
(D2)                                  TRUE
```

Function: scopy (string)

Returns a copy of `string` as a new string.

Function: sdowncase (string)
Function: sdowncase (string,start)
Function: sdowncase (string,start,end)

Like `supcase`, but uppercase characters are converted to lowercase.

Function: sequal (string1,string2)

Returns `true` if `string1` and `string2` are the same length and contain the same characters.

Function: sequalignore (string1,string2)

Like `clessp` but ignores case.

Function: sexplode (string)

`sexplode` is an alias for function `charlist`.

Function: simplode (list)
Function: simplode (list,delim)

`simplode` takes a list of expressions and concatenates them into a string. If no delimiter `delim` is used, `simplode` is like `sconc` and uses no delimiter. `delim` can be any string.

```
(C1) simplode(["xx[",3,"]:",expand((x+y)^3)]);
(D1)                    xx[3]:y^3+3*x*y^2+3*x^2*y+x^3
(C2) simplode( sexplode("stars")," * " );
(D2)                           s * t * a * r * s
(C3) simplode( ["One","more","coffee."]," " );
(D3)                           One more coffee.
```

Function: sinsert (seq,string,pos)
Returns a string that is a concatenation of substring(string,1,pos-1), the string seq and substring (string,pos). Note that the first character in string is in position 1.

```
(C1) s: "A submarine."$
(C2) sconc( substring(s,1,3),"yellow ",substring(s,3) );
(D2)                          A yellow submarine.
(C3) sinsert("hollow ",s,3);
(D3)                          A hollow submarine.
```

Function: slength (string)
Returns the number of characters in string.

Function: smake (num,char)
Returns a new string with a number of num characters char.

```
(C1) smake(3,"w");
(D1)                                   www
```

Function: smismatch (string1,string2)
Function: smismatch (string1,string2,test)
Returns the position of the first character of string1 at which string1 and string2 differ or false. Default test function for matching is cequal. If smismatch should ignore case, use cequalignore as test.

Function: split (string)
Function: split (string,delim)
Function: split (string,delim,multiple)
Returns the list of all tokens in string. Each token is an unparsed string. split uses delim as delimiter. If delim is not given, the space character is the default delimiter. multiple is a boolean variable with true by default. Multiple delimiters are read as one. This is useful if tabs are saved as multiple space characters. If multiple is set to false, each delimiter is noted.

```
(C1) split("1.2   2.3   3.4   4.5");
(D1)                         [1.2, 2.3, 3.4, 4.5]
(C2) split("first;;third;fourth",";",false);
(D2)                     [first, , third, fourth]
```

Function: sposition (char,string)
Returns the position of the first character in string which matches char. The first character in string is in position 1. For matching characters ignoring case see ssearch.

Function: sremove (seq,string)
Function: sremove (seq,string,test)
Function: sremove (seq,string,test,start)
Function: sremove (seq,string,test,start,end)
Returns a string like string but without all substrings matching seq. Default test function for matching is cequal. If sremove should ignore case while searching for seq, use cequalignore as test.
Use start and end to limit searching. Note that the first character in string is in position 1.

```
(C1) sremove("n't","I don't like coffee.");
(D1)                          I do like coffee.
(C2) sremove ("DO ",%,'cequalignore);
(D2)                           I like coffee.
```

Function: sremovefirst (seq,string)
Function: sremovefirst (seq,string,test)
Function: sremovefirst (seq,string,test,start)
Function: sremovefirst (seq,string,test,start,end)
Like sremove except that only the first substring that matches seq is removed.

Function: sreverse (string)
Returns a string with all the characters of string in reverse order.

```
Function: ssearch (seq,string)
Function: ssearch (seq,string,test)
Function: ssearch (seq,string,test,start)
Function: ssearch (seq,string,test,start,end)
```

Returns the position of the first substring of `string` that matches the string `seq`. Default test function for matching is `cequal`. If `ssearch` should ignore case, use `cequalignore` as `test`. Use `start` and `end` to limit searching. Note that the first character in `string` is in position 1.

```
(C1) ssearch("~s","~{~S ~}~%",'cequalignore);
(D1)                                        3
```

```
Function: ssort (string)
Function: ssort (string,test)
```

Returns a string that contains all characters from `string` in an order such there are no two successive characters `c` and `d` such that `test(c,d)` is false and `test(d,c)` is true. Default test function for sorting is `clessp`. The set of test functions is { `clessp`, `clesspignore`, `cgreaterp`, `cgreaterpignore`, `cequal`, `cequalignore` }.

```
(C1) ssort("I don't like Mondays.");
(D1)                              '.IMaddeiklnnoosty
(C2) ssort("I don't like Mondays.",'cgreaterpignore);
(D2)                           ytsoonnMlkIiedda.'
```

```
Function: ssubst (new,old,string)
Function: ssubst (new,old,string,test)
Function: ssubst (new,old,string,test,start)
Function: ssubst (new,old,string,test,start,end)
```

Returns a string like `string` except that all substrings matching `old` are replaced by `new`. `old` and `new` need not to be of the same length. Default test function for matching is `cequal`. If `ssubst` should ignore case while searching for `old`, use `cequalignore` as `test`.

Use `start` and `end` to limit searching. Note that the first character in `string` is in position 1.

```
(C1) ssubst("like","hate","I hate Thai food. I hate green tea.");
(D1)                  I like Thai food. I like green tea.
(C2) ssubst("Indian","thai",%,'cequalignore,8,12);
(D2)                 I like Indian food. I like green tea.
```

```
Function: ssubstfirst (new,old,string)
Function: ssubstfirst (new,old,string,test)
Function: ssubstfirst (new,old,string,test,start)
Function: ssubstfirst (new,old,string,test,start,end)
```

Like `subst` except that only the first substring that matches `old` is replaced.

```
Function: strim (seq,string)
```

Returns a string like `string` except that all substrings matching `old` are replaced by `new`. `old` and `new` need not to be of the same length. Default test function for matching is `cequal`. If `ssubst` should ignore case while searching for `old`, use `cequalignore` as `test`.

Use `start` and `end` to limit searching. Note that the first character in `string` is in position 1.

```
(C1) "/* comment */"$
(C2) strim(" /*",%);
(D2)                                  comment
(C3) slength(%);
(D3)                                        7
```

```
Function: striml (seq,string)
```

Like `strim` except that only the left end of `string` is trimmed.

```
Function: strimr (seq,string)
```

Like `strim` except that only the right end of `string` is trimmed.

Function: substring (string,start)
Function: substring (string,start,end)

Returns the substring of `string` beginning at position `start` and ending at position `end`. The character at position `end` is not included. If `end` is not given, the substring contains the rest of the string. Note that the first character in `string` is in position 1.

```
(C1) substring("substring",4);
(D1)                                  string
(C2) substring(%,4,6);
(D2)                                    in
```

Function: supcase (string)
Function: supcase (string,start)
Function: supcase (string,start,end)

Returns `string` except that lowercase characters from position `start` to `end` are replaced by the corresponding uppercase ones. If `end` is not given, all lowercase characters from `start` to the end of `string` are replaced.

```
(C1) supcase("english",1,2);
(D1)                                  English
```

Function: tokens (string)
Function: tokens (string,test)

Returns a list of tokens, which have been extracted from `string`. The tokens are substrings whose characters satisfy a certain test function. If `test` is not given, `constituent` is used as the default test. {`constituent, alphacharp, digitcharp, lowercasep, uppercasep, charp, characterp, alphanumericp`} is the set of test functions.

(The Lisp-version of `tokens` is written by Paul Graham. ANSI Common Lisp, 1996, page 67.)

```
(C1) tokens("24 October 2005");
(D1)                             [24, October, 2005]
(C2) tokens("05-10-24",'digitcharp);
(D2)                               [05, 10, 24]
(C3) map(parsetoken,%);
(D3)                                [5, 10, 24]
```