Strings and characters

The following string and character functions will be of use in the course. Further information on these and other functions can be found in the DrScheme Help Desk (see the "Helpful Tips" page on the course Web site for information about the Help Desk).

1 Strings

A string is a data type. We use double quotation marks as the marker at the beginning and the end to signal to DrScheme that it is a string instead of, for example, the name of a function. You will type quotes to indicate that you are referring to a string, and DrScheme will display quotes to indicate that what is displayed is a string.

All of the following examples are strings:

```
"This is a string."
"Agent 007"
"@#&%!"
```

The function application (*string? x*) determines whether *x* (which can be any type of data) is a string. For example,

```
(string? 5) ; evaluates to false
(string? "This is a string.") ; evaluates to true
```

The function *string-length* determines the number of characters in a string. As discussed in Section 2, characters can be letters, numbers, or punctuation signs. A blank is also considered a character; (*string-length* `"This is a string."`) evaluates to 17 as there are thirteen letters, three blank spaces, and one punctuation mark.

```
;; string?: any → boolean
;; string-length: string → nat
```

1.1 Forming strings from numbers

The function *number->string* produces a string from a number. For example, (*number->string* 5) produces the string `"5"`. Note that (*string?* (*number->string* 5)) evaluates to true.

1.2 Forming strings from other strings

We can form strings from other strings using the functions *string-append* and *substring*. The function *string-append* consumes one or more strings and produces the string obtained by gluing them together in order. For example,

(*string-append* `"now" "here"`)

evaluates to "nowhere". The function application

(*substring str start end*)

evaluates to the string that is formed from the symbols starting at position *start* and ending right before the position *end*, where the first character in a string is in position zero. For example, (*substring* "caterpillar" 5 9) evaluates to "pill". Note that the difference between indices gives you the number of symbols in the substring.

1.3 Comparing strings

Strings can be compared with each other, where *string1* is less than *string2* if *string1* comes before *string2* in alphabetical order. Each of the following functions consume two strings as input and produce a Boolean value:

;; string=? : string string → boolean
;; string<? : string string → boolean
;; string>? : string string → boolean
;; string<=? : string string → boolean
;; string>=? : string string → boolean

All capital letters, in alphabetical order, come before all lower-case letters, in alphabetical order, with blanks coming before all letters. More details on order are listed in the section on characters, below.

Here are some examples:

(*string=?* "first" "second") ; yields false
(*string<?* "Zebra" "antelope") ; yields true, since Z comes before a.
(*string>?* "isn't" "is not") ; yields true, since n comes after blank.

The functions *string<=?* and *string>=?* are defined analogously.

2 Characters

Characters are another data type. For characters, we use #\ as the marker at the beginning to signal to DrScheme that it is a character. As with strings, you will type in the marker to indicate that what follows is a character, and DrScheme will display the marker to indicate that what follows is a character.

All of the following examples are characters:

#\a
#\space
#\5
#\,
#\J

There are functions to determine whether a value is a character and whether a character is a lower-case letter, an upper-case letter, a digit, or a blank space.

```
;; char? : any → boolean
;; char-lower-case? : char → boolean
;; char-upper-case? : char → boolean
;; char-numeric? : char → boolean
;; char-whitespace? : char → boolean
```

All of the following evaluate to true.

```
(char? #\,)
(char-lower-case? #\a)
(char-upper-case? #\A)
(char-numeric? #\1)
(char-whitespace? #\space)
```

The order of characters can be determined using the functions below, where characters are in the following order: blank space, punctuation marks, digits, upper-case letters, lower-case letters. Upper-case letters are in alphabetical order, lower-case letters are in alphabetical order, and digits are in numerical order.

Each of the following functions consumes two or more characters; you can view the question as being asked about each consecutive pair of characters in the input. The "ci" indicates that the comparison is case-insensitive, as if lower-case and upper-case letters were the same.

```
;; char<=? : char char ... → boolean
;; char<? : char char ... → boolean
;; char=? : char char ... → boolean
;; char>=? : char char ... → boolean
;; char>? : char char ... → boolean
;; char-ci<=? : char char ... → boolean
;; char-ci<? : char char ... → boolean
;; char-ci=? : char char ... → boolean
;; char-ci>=? : char char ... → boolean
;; char-ci>? : char char ... → boolean
```

Example:

```
(char<? #\, #\1 #\A #\a) ; evaluates to true
```

The following functions modify characters to their lower-case or upper-case versions. There is no change if a character is already a lower-case (respectively, upper-case) letter, or if it is a number, space, or punctuation mark.

```
;; char-downcase : char → char
;; char-upcase : char → char
```

The following function can be used to extract from a string the character in the specified position.

;; string-ref: string nat → char

For example (*string-ref* `"string" 1`) yields `#\t`.

We also make use of the following functions to convert between strings and lists of characters.

;; list->string : (listof char) → string
;; string->list : string → (listof char)