Chapter 11

Character Strings

Data structures, such as vectors, matrices and arrays, have elements that have thus far been assumed to be either numeric or complex values. This chapter introduces a third possibility for these elements: character strings.

A single character in R is stored in memory in eight consecutive bits, which computer scientists call a "byte." A *bit* is a binary digit. There are $2^8 = 256$ different sequences of eight bits, and the meaning of these 256 possibilities are defined by the American Standard Code for Information Interchange, which is abbreviated by ASCII. The 26 lower-case letters, the 26 upper-case letters, and dozens of symbols are defined as particular strings of eight bits in ASCII. A single such letter or symbol is known as a "character." A sequence of characters is known as a "character string."

This chapter introduces character strings in R. The topics considered in this chapter are (a) simple character strings, (b) automatic coercion, (c) built-in strings, (d) string manipulation, (e) names, and (f) factors.

11.1 Simple character strings

Placing one or more characters in single or double quotes constitutes a character string. In order to assign the string `"R"` to the object `x` and display the value of `x`, type

```
> x = "R"                                  # a character: ASCII 01010010
> x                                        # display x
[1] "R"
```

The `[1]` is again displayed because `x` is a vector of length one consisting of a single element that has been assigned the string `"R"`. Using single quotes to assign the string `'S'` to `x` is done in the same fashion:

```
> x = 'S'                                  # a character: ASCII 01010011
> x                                        # display x
[1] "S"
> nchar(x)                                 # character count
[1] 1
```

R displays strings surrounded by double quotes, regardless of whether they are entered with single or double quotes. The `nchar` function accepts a string as its argument and returns the number of

characters in the string, which in this case is one. Strings with multiple characters are input in the same fashion:

```
> x = "Hello world!"                    # a string of characters
> x                                     # display x
[1] "Hello world!"
> nchar(x)                              # character count
[1] 12
```

This time the `nchar` function returns 12, which is a count of the characters in the string `"Hello world!"` (not including the quotes, but including the space and the exclamation point). The `substr` (substring) function allows a portion of a string to be extracted.

```
> substr(x, start = 2, stop = 10)       # substring function
[1] "ello worl"
```

Occasions might arise in which you would like to have a quotation mark or apostrophe within a character string. Placing a backslash before the quotation mark or apostrophe is the way to accomplish this. When `nchar` is called in the example below, it counts the apostrophe as just a single character.

```
> x = "I can\'t go there"               # a string of characters
> x                                     # display x
[1] "I can't go there"
> nchar(x)                              # character count
[1] 16
```

Octal (base 8) is a compact way to write a binary number. Beginning a string with a backslash indicates an octal number in a string. The octal number 007 (binary 000000111) corresponds to the chime sound on a laptop or desktop computer. So one way to sound this chime is to use the `cat` (concatenate) function to output this particular octal code:

```
> x = "\007"                            # bell sound as octal ASCII code
> cat(x)                                # ring the bell
```

The upper-case R is binary code 001010010 in ASCII, which is octal 122. Calling the `print` function below prints `R` as a one-element vector that consists of a character string as an element, but calling the `cat` function just prints the letter R (not as a vector and without the quotation marks).

```
> x = "\122"                            # letter R as octal ASCII code
> print(x)                              # prints the object x
[1] "R"
> cat(x)                                # display R
R
```

Finally, calling the `cat` function to generate a tab, followed by the letter R, followed by a newline results in

```
> cat("\t", x, "\n")                    # tab then display R then newline
   R
```

11.2 Automatic coercion

R does not allow you to mix data types within the data structures that have been introduced so far. For example, you cannot mix numeric values, complex values, and strings together in a vector. (There is a data structure known as a *list*, introduced in Chapter 15, which allows this mixing.) If you do mix these data types, R will perform what is known as "automatic coercion," and it will coerce the elements to all have the same type. In setting `x` to a vector of strings and numeric values, for example, R will coerce all values to be strings:

```
> x = c("R", 4.7, 3, -8, 6.02e23,  pi)     # mixed data types; auto. coercion
> x                                         # display x
[1] "R"                "4.7"              "3"                "-8"
[5] "6.02e+23"         "3.14159265358979"
```

Coercing numerics to be character strings is more sensible than coercing character strings to be numerics (for example, it is not clear what numerical value should be assigned to `"Hello world!"`). Applying the `nchar` function to the vector `x` results in the following six counts:

```
> nchar(x)                                  # character count
[1]  1  3  1  2  8 16
```

Applying the `substr` function with `start = 2` and `stop = 4` (the arguments are referenced by position here) to the `x` vector results in

```
> substr(x, 2, 4)                           # substring function
[1] ""    ".7"  ""    "8"   ".02" ".14"
```

11.3 Built-in objects of character strings

R contains some built-in objects consisting of character strings as elements that can often be useful as labels in graphics. The objects `letters` and `LETTERS`, for instance, are each 26-element vectors that contain the lower-case and upper-case letters.

```
> letters                                   # lower-case letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r"
[19] "s" "t" "u" "v" "w" "x" "y" "z"
> LETTERS                                   # upper-case letters
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R"
[19] "S" "T" "U" "V" "W" "X" "Y" "Z"
```

A specific letter can be accessed using a subscript in the usual fashion:

```
> LETTERS[18]                               # one upper-case letter
[1] "R"
```

Two other vectors of built-in strings are `month.name` and `month.abb`, which contain the months of the year and their abbreviations.

```
> month.name                                # month names
 [1] "January"   "February"  "March"     "April"     "May"       "June"
 [7] "July"      "August"    "September" "October"   "November"  "December"
> month.abb                                 # month abbreviations
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```