

Character Values in Scheme

Summary: We consider *characters*, one of the important primitive data types in many languages. Characters are the building blocks of strings (which we cover in a subsequent reading).

Procedures and Constants Covered in This Reading:

- Constant notation: `#\ch` (character constants) `"string"` (string constants).
- Character constants: `#\a` (lowercase a) ... `#\z` (lowercase z); `#\A` (uppercase A) ... `#\Z` (uppercase Z); `#\0` (zero) ... `#\9` (nine); `#\space` (space); `#\newline` (newline); and `#\?` (question mark).
- Character conversion: `char->integer`, `integer->char`, `char-downcase`, and `char-upcase`
- Unary character predicates: `char?`, `char-alphabetic?`, `char-numeric?`, `char-lower-case?`, `char-upper-case?`, and `char-whitespace?`.
- Character comparison predicates (warning: reference links do not yet work): `char<?`, `char<=?`, `char=?`, `char>=?`, `char>?`, `char-ci<?`, `char-ci<=?`, `char-ci=?`, `char-ci>=?`, and `char-ci>?`.

Contents:

- Introduction
- Characters in Scheme
- Collating Sequences
- Handling Case
- More Character Predicates
- Appendix: Representing Characters
 - ASCII
 - Unicode

Introduction

A *character* is a small, repeatable unit within some system of writing -- a letter or a punctuation mark, if the system is alphabetic, or an ideogram in a writing system like Han (Chinese). Characters are usually put together in sequences called *strings*.

Although early computer programs focused primarily on numeric processing, as computation advanced, it grew to incorporate a variety of algorithms that incorporated characters and strings. Some of the more interesting algorithms we will consider involve these data types. Hence, we must learn how to use this building blocks.

Characters in Scheme

As you might expect, Scheme needs a way to distinguish between many different but similar things, including: characters (the units of writing), strings (formed by combining characters), symbols (which are treated as atomic and also cannot be combined or separated), and variables (names of values). Similarly, Scheme needs to distinguish between numbers (which you can compute with) and digit characters (which you can put in strings).

In Scheme, a name for any of the text characters can be formed by writing `>#\` before that character. For instance, the expression `#\a` denotes the lower-case a (to be distinguished from the symbol that you obtain with `'a`, from the name `a`, and from the string `"a"`), the expression `#\3` denotes the character 3 (to be distinguished from the number 3) and the expression `#\?` denotes the question mark (to be distinguished from a symbol and a name that look quite similar). In addition, the expression `#\space` denotes the space character, and `#\newline` denotes the newline character (the one that is used to terminate lines of text files stored on Unix and Linux systems).

Collating Sequences

In any implementation of Scheme, it is assumed that the available characters can be arranged in sequential order (the “collating sequence” for the character set), and each character is associated with an integer that specifies its position in that sequence. In ASCII, the numbers that are associated with characters run from 0 to 127; in Unicode, they lie within the range from 0 to 65535. (Fortunately, Unicode includes all of the ASCII characters and associates with each one the same collating-sequence number that ASCII uses.) Applying the built-in `char->integer` procedure to a character gives you the collating-sequence number for that character; applying the converse procedure, `integer->char`, to an integer in the appropriate range gives you the character that has that collating-sequence number.

The importance of the collating-sequence numbers is that they extend the notion of alphabetical order to all the characters. Scheme provides five built-in predicates for comparing characters (`char<?`, `char<=?`, `char=?`, `char>=?`, and `char>?`). They all work by determining which of the two characters comes first in the collating sequence (that is, which one has the lower collating-sequence number).

Scheme requires that if you compare two capital letters or two lower-case letters, you'll get standard alphabetical order: `(char<? #\A #\Z)` must be true, for instance. If you compare a capital letter with a lower-case letter, though, the result depends on the design of the character set. In ASCII, every capital letter -- even `#\Z` -- precedes every lower-case letter -- even `#\a`. Similarly, if you compare two digit characters, Scheme guarantees that the results will be consistent with numerical order: `#\0` precedes `#\1`, which precedes `#\2`, and so on. But if you compare a digit with a letter, or anything with a punctuation mark, the results depend on the character set.

Handling Case

Because there are many applications in which it is helpful to ignore the distinction between a capital letter and its lower-case equivalent in comparisons, Scheme also provides *case-insensitive* versions of the comparison procedures: `char-ci<?`, `char-ci<=?`, `char-ci=?`, `char-ci>=?`, and `char-ci>?`. These procedures essentially convert all letters to the same case (in DrScheme, upper case) before

comparing them.

There are also two procedures for converting case.

- If its argument is a lower-case letter, `char-upcase` returns the corresponding capital letter; otherwise, it returns the argument unchanged.
- If its argument is a capital letter, `char-downcase` returns the corresponding lower-case letter; otherwise, it returns the argument unchanged.

More Character Predicates

Scheme provides several one-argument predicates that apply to characters:

- `char-alphabetic?` determines whether its argument is a letter (`#\a` through `#\Z` or `#\A` through `#\Z`, in English).
- `char-numeric?` determines whether its argument is a digit character (`#\0` through `#\9` in our standard base-ten numbering system).
- `char-whitespace?` determines whether its argument is a “whitespace character”, one that is conventionally stored in a text file primarily to position text legibly. In ASCII, the whitespace characters are the space character and four specific control characters: `<Control/I>` (tab), `<Control/J>` (line feed), `<Control/L>` (form feed), and `<Control/M>` (carriage return). On most systems, `#\newline` is a whitespace character. On our Linux systems, `#\newline` is the same as `<Control/J>` and so counts as a whitespace character.
- `char-upper-case?` determines whether its argument is a capital letter.
- `char-lower-case?` determines whether its argument is a lower-case letter.

It may seem that it’s easy to implement some of these operations. For example, you might want to implement `char-alphabetic?` as

A character is alphabetic if it is between `#\a` through `#\Z` or between `#\A` through `#\Z`

However, that implementation is not necessarily correct for all versions of Scheme: Since Scheme does not guarantee that the letters are collated without gaps, it’s possible that this algorithm treats some non-letters as letters. The alternative, comparing to each valid letter in turn, seems inefficient. By making this procedure built-in, the designers of Scheme have encouraged programmers to rely on a correct (and, presumably, efficient) implementation.

Note that all of these predicates assume that their parameter is a character. Hence, if you don’t know the type of a parameter, you will need to first ensure that it is a character. You will learn to do combine tests when we explore conditionals.

Appendix: Representing Characters

When a character is stored in a computer, it must be represented as a sequence of *bits* (“binary digits”, that is, zeroes and ones). However, the choice of a particular bit sequence to represent a particular character is more or less arbitrary. In the early days of computing, each equipment manufacturer developed one or more “character codes” of its own, so that, for example, the capital letter A was represented by the

sequence 110001 on an IBM 1401 computer, by 000001 on a Control Data 6600, by 11000001 on an IBM 360, and so on. This made it troublesome to transfer character data from one computer to another, since it was necessary to convert each character from the source machine's encoding to the target machine's encoding. The difficulty was compounded by the fact that different manufacturers supported different characters; all provided the twenty-six capital letters used in writing English and the ten digits used in writing Arabic numerals, but there was much variation in the selection of mathematical symbols, punctuation marks, etc.

ASCII

In 1963, a number of manufacturers agreed to use the American Standard Code for Information Interchange (ASCII), which is currently the most common and widely used character code. It includes representations for ninety-four characters selected from American and Western European text, commercial, and technical scripts: the twenty-six English letters in both upper and lower case, the ten digits, and a miscellaneous selection of punctuation marks, mathematical symbols, commercial symbols, and diacritical marks. (These ninety-four characters are the ones that can be generated by using the forty-seven lighter-colored keys in the typewriter-like part of a MathLAN workstation's keyboard, with or without the simultaneous use of the <Shift> key.) ASCII also reserves a bit sequence for a "space" character, and thirty-three bit sequences for so-called *control characters*, which have various implementation-dependent effects on printing and display devices -- the "newline" character that drops the cursor or printing head to the next line, the "bell" or "alert" character that causes the workstation to beep briefly, and such like.

In ASCII, each character or control character is represented by a sequence of exactly seven bits, and every sequence of seven bits represents a different character or control character. There are therefore 2^7 (that is, 128) ASCII characters altogether.

Unicode

Over the last quarter-century, non-English-speaking computer users have grown increasingly impatient with the fact that ASCII does not provide many of the characters that are essential in writing other languages. A more recently devised character code, the Unicode Worldwide Character Standard, currently defines bit sequences for 49194 characters for the Arabic, Armenian, Bengali, Bopomofo, Canadian Syllabics, Cherokee, Cyrillic, Devanagari, Ethiopic, Georgian, Greek, Gujarati, Gurmukhi, Han, Hangul, Hebrew, Hiragana, Kannada, Katakana, Khmer, Latin, Lao, Malayalam, Mongolian, Myanmar, Ogham, Oriya, Runic, Sinhala, Tamil, Telugu, Thaana, Thai, Tibetan, and Yi writing systems, as well as a large number of miscellaneous numerical, mathematical, musical, astronomical, religious, technical, and printers' symbols, components of diagrams, and geometric shapes.

Unicode uses a sequence of sixteen bits for each character, allowing for 2^{16} (that is, 65536) codes altogether. Many bit sequences are still unassigned and may, in future versions of Unicode, be allocated for some of the numerous writing systems that are not yet supported. The designers have completed work on the Deseret, Etruscan, and Gothic writing systems, although they have not yet been added to the Unicode standard. Characters for the Shavian, Linear B, Cypriot, Tagalog, Hanunóo, Buhid, Tagbanwa, Cham, Tai, Glagolitic, Coptic, Buginese, Old Hungarian Runic, Phoenician, Avestan, Tifinagh, Javanese, Rong, Egyptian Hieroglyphic, Meroitic, Old Persian Cuneiform, Ugaritic Cuneiform, Tengwar, Cirth, tlhIngan Hol (i.e., "Klingon¹"), Brahmi, Old Permic, Sinaitic, South Arabian, Pollard, Blissymbolics, and

Soyombo writing systems are under consideration or in preparation.

Although our local Scheme implementations use and presuppose the ASCII character set, the Scheme language does not require this, and Scheme programmers should try to write their programs in such a way that they could easily be adapted for use with other character sets (particularly Unicode).

Endnotes

¹ Can you tell that CS folks are geeks?

Copyright © 2006 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Endnotes

¹ Can you tell that CS folks are geeks?