Lecture 3:
Strings

Steven Skiena

Department of Computer Science
State University of New York
Stony Brook, NY 11794–4400

http://www.cs.sunysb.edu/∼skiena

Character Codes

Character codes are mappings between numbers and the symbols which make up a particular alphabet.
The *American Standard Code for Information Interchange* (ASCII) is a single-byte character code where $2^7 = 128$ characters are specified. Bytes are eight-bit entities; so that means the highest-order bit is left as zero.

0	NUL	1	SOH	2	STX	3	ETX	4	EOT	5	ENQ	6	ACK	7	BEL
8	BS	9	HT	10	NL	11	VT	12	NP	13	CR	14	SO	15	SI
16	DLE	17	DC1	18	DC2	19	DC3	20	DC4	21	NAK	22	SYN	23	ETB
24	CAN	25	EM	26	SUB	27	ESC	28	FS	29	GS	30	RS	31	US
32	SP	33	!	34	”	35	#	36	$	37	%	38	&	39	’
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	/	93	]	94	ˆ	95	_
96	‘	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	—	125	}	126	∼	127	DEL

Properties of ASCII

Several properties of the design make programming tasks easier:

- All non-printable characters have either the first three bits as zero or all seven lowest bits as one. This makes it very easy to eliminate them before displaying junk.
- Both the upper- and lowercase letters and the numerical digits appear sequentially. Thus we can iterate through all the letters/digits simply by looping from the value of the first symbol (say, “a”) to value of the last symbol (say, “z”).

- We can convert a character (say, “`I`”) to its rank in the collating sequence (eighth, if “`A`” is the zeroth character) simply by subtracting off the first symbol (“`A`”).

- We can convert (say “`C`”) from upper- to lowercase by adding the difference of the upper and lowercase starting character (“`C`”-“`A`”+“`a`”). Similarly, a character x is uppercase if and only if it lies between “`A` ” and “`Z`”.

- The character code tells us what will happen when naively sorting text files. Which of “`x`” or “`3`” or “`C`” appears first in alphabetical order? Sorting alphabetically means sorting by character code. Using a different collating sequence requires more complicated comparison functions.

- Non-printable character codes for new-line (10) and carriage return (13) are designed to delimit the end of text lines. Inconsistent use of these codes is one of the pains in moving text files between UNIX and Windows systems.

Unicode

More modern international character code designs such as *Unicode* use two or even three bytes per symbol, and can represent virtually any symbol in every language on earth.

Older languages, like Pascal, C, and C++, view the `char` type as virtually synonymous with 8-bit entities. However, good old ASCII remains alive embedded in Unicode.

Java, on the other hand, was designed to support Unicode, so characters are 16-bit entities. The upper byte is all zeros when working with ASCII/ISO Latin 1 text.

Representing Strings

Strings are sequences of characters, where order clearly matters. It is important to be aware of how your favorite programming language represents strings, because there are several different possibilities:

- *Null-terminated Arrays* – C/C++ treats strings as arrays of characters. The string ends the instant it hits the null character "\0", i.e., zero ASCII. Failing to end your string explicitly with a null typically extends it by a bunch of unprintable characters.

- *Array Plus Length* – Another scheme uses the first array location to store the length of the string, thus avoiding the need for any terminating null character. Presumably this is what Java implementations do internally.
- *Linked Lists of Characters* – Text strings can be represented using linked lists, but this is typically avoided because of the high space-overhead associated with having a several-byte pointer for each single byte character.

Which String Representation?

The underlying string representation can have a big impact on which operations are easily or efficiently supported. Compare each of these three data structures with respect to the following properties:

- Which uses the least amount of space? On what sized strings?
- Which constrains the content of the strings which can possibly be represented?
- Which allow constant-time access to the ith character?

- Which allow efficient checks that the ith character is in fact within the string, thus avoiding out-of-bounds errors?
- Which allow efficient deletion or insertion of new characters at the ith position?
- Which representation is used when users are limited to strings of length at most 255, e.g., file names in Windows?

Searching for Patterns

The simplest algorithm to search for the pattern string p in text t overlays the pattern string on the text, and checks whether every pattern character matches the corresponding text character:

```
/*      Return position of the first occurrence of pattern
        p in the text t, and -1 if it does not occur.
*/

int findmatch(char *p, char *t)
{
        int i,j;                  /* counters */
        int plen, tlen;           /* string lengths */

        plen = strlen(p);
        tlen = strlen(t);

        for (i=0; i<=(tlen-plen); i=i+1) {
                j=0;
                while ((j<plen) && (t[i+j]==p[j]))
                        j = j+1;
                if (j == plen) return(i);
        }

        return(-1);
}
```

Note that this routine only searches for exact pattern matches. If a letter is capitalized in the pattern but not in the text there is no match.

This algorithm runs in $O(|p| \times |q|)$ time. More complicated but efficient *linear-time* algorithms exist for substring pattern matching.

C String Library Functions

The C language *character* library `ctype.h` contains several simple tests and manipulations on character codes. As with all C predicates, true is defined as any non-zero quantity, and false as zero.

```
#include <ctype.h>       /* include the character library */

int isalpha(int c);      /* true if c is either upper or lower case */
int isupper(int c);      /* true if c is upper case */
int islower(int c);      /* true if c is lower case */
int isdigit(int c);      /* true if c is a numerical digit (0-9) */
int ispunct(int c);      /* true if c is a punctuation symbol */
int isxdigit(int c);     /* true if c is a hexadecimal digit (0-9,A-F) */
int isprint(int c);      /* true if c is any printable character */

int toupper(int c);      /* convert c to upper case -- no error checking */
int tolower(int c);      /* convert c to lower case -- no error checking */
```

These appear in the C language *string* library `string.h`.

```
#include <string.h>      /* include the string library */

char *strcat(char *dst, const char *src);     /* concatenation */
int strcmp(const char *s1, const char *s2);   /* is s1 == s2? */
char *strcpy(char *dst, const char *src);     /* copy src to dist */
size_t strlen(const char *s);                 /* length of string */
char *strstr(const char *s1, const char *s2); /* search for s2 in s1 */
char *strtok(char *s1, const char *s2);       /* iterate words in s1 */
```

C++ String Library Functions

In addition to supporting C-style strings, C++ has a string class which contains methods for these operations and more:

```
string::size()          /* string length */
string::empty()         /* is it empty */

string::c_str()         /* return a pointer to a C style string */

string::operator [](size_type i)        /* access the ith character */

string::append(s)       /* append to string */
string::erase(n,m)      /* delete a run of characters */
string::insert(size_type n, const string&s) /* insert string s at n */

string::find(s)
string::rfind(s)        /* search left or right for the given string */

string::first()
string::last()          /* get characters, also there are iterators */
```

Overloaded operators exist for concatenation and string comparison.

Java String Objects

Java strings are first-class objects deriving either from the `String` class or the `StringBuffer` class. The `String` class is for static strings which do not change, while `StringBuffer` is designed for dynamic strings.

Recall that Java was designed to support Unicode, so its characters are 16-bit entities.

The `java.text` package contains more advanced operations on strings, including routines to parse dates and other structured text.

110302 (Where's Waldorf)

Find words in a grid a letters.
What is the easiest way to write a comparison function for all eight directions?

110304 (Crypt Kicker II)

Solve a substition cipher via a known plain text attack.

How do we identify what the plaintext sentence is?

110306 (File Fragmentation)

Put together a collection of broken copies of a given text string.

Which pair of fragments go together?

How can we find the right order of the pair?

110307 (Doublets)

Build word ladders on a dictionary of strings.
How do we represent and traverse the underlying graph? (if necessary, look ahead to Chapter 9)