Lecture 3
Strings

Euiseong Seo
(euiseong@skku.edu)

Strings

- **One of the fundamental data structures**
  - Internet search engines
  - Computational biology
  - Processing multimedia data
  - Big data analysis
- **One of the plain data structures to manipulate**
  - Usually linear
  - Easy to be parallelized
- **In C, a string is an array of characters + null**

Character Code

- **Character codes are mappings between numbers and symbols, which make up a particular alphabet**

- **ASCII code**
  - American Standard Code for Information Exchange
  - 7 bit code, thus 2^7 characters can be represented
  - The highest order bit is always left as zero
  - De-facto-standard for alpha-numeric character representation

ASCII Table

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	^	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 easier**
  - First three bits are zeros or all seven bits are ones = non-printable characters
  - Both the upper- and lowercase letters and the numerical digits appear sequentially
  - We can convert a character (say, “I”) to its rank in the collating sequence
  - We can convert (say “C”) from upper- to lowercase by a dding the difference of the upper and lowercase starting character (“C”-“A”+“a”)

Properties of ASCII

- 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
- Non-printable character codes for new-line (10) and carriage return (13) are designed to delimit the end of text lines
  - LF+CR vs. LF

Unicode

- **Two or three bytes for a symbol**
- **For any symbol in every language on earth**
- **ASCII remains alive embedded in Unicode**
  - Programs written in old languages can run correctly
  - The upper bytes are all zeros for ASCII/Latin-1
- **Java uses 16-bit Unicode representation**
  - A character is always 16-bit long

String Representations

- **Strings are sequences of characters**
  - Order clearly matters
- **Several different representations**
  - Null-terminated arrays
    - C or C++ default
    - Failing to terminate a string with null typically extends it by a bunch of unprintable characters
  - Array + length
    - Java
    - No need for null character
  - Linked list of characters
    - High space overhead
    - Easy substring replacement

String Representation Choice

- **A big impact on which operations are easily or efficiently supported**
- **Points to consider**
  - Which uses the least amount of space?
    On what sized strings?
  - Which allow constant-time access to the i-th character?
  - Which allow efficient checks that the i-th character is in fact within the string?
  - Which allow efficient deletion or insertion of new charac ters at the ith position?
  - Which representation is used when users are limited to strings of length at most 255?

String Libraries

- Many languages provide string manipulation packages
- Don’t reinvent the wheel

String Libraries for C

```
#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 */
```

String Libraries for C

```
#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 */
```

String Class for C

```
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 */
```

String Class for Java

- Strings are first-class objects deriving either from the *String* class or the *StringBuffer* class
- String for static string
- StringBuffer for dynamic string

Problem Example: Corporate Renaming

- **Company names frequently change**
  - For bankruptcy
  - To change notorious image
  - To increase stock price
  - For M&A
- **Examples**
  - Anderson Consulting to Accenture
  - Enron to Dynegy
  - DEC to Compaq
  - TWA to American
- **Our goal is to replace the original names in a text to the new ones**

Input/Output Example

```
4
"Anderson Consulting" to "Accenture"
"Enron" to "Dynegy"
"DEC" to "Compaq"
"TWA" to "American"
5
Anderson Accounting begat Anderson Consulting, which
offered advice to Enron before it DECLARED bankruptcy,
which made Anderson
Consulting quite happy it changed its name
in the first place!
```

Which should be transformed to —

```
Anderson Accounting begat Accenture, which
offered advice to Dynegy before it CompaqLARED bankruptcy,
which made Anderson
Consulting quite happy it changed its name
in the first place!
```

Your Plan

- **Read the M&A list into a DB**
  - define DB structure
  - how to handle the double quote (“) sign?
- **main loop**
  - compare each line of a doc with each DB entry
  - if there is a match, replace it with a new name
- **Now, define**
  - functions
  - global variables

Data Structure

```
#include <string.h>

#define MAXLEN           1001    /* longest possible string */
#define MAXCHANGES       101     /* maximum number of name changes */

typedef char string[MAXLEN];

string mergers[MAXCHANGES][2];   /* store before/after corporate names */
int nmergers;                    /* number of different name changes */
```

Read Name Changes

```
read_changes()
{
        int i;                  /* counter */

        scanf("%d\n",&nmergers);
        for (i=0; i<nmergers; i++) {
                read_quoted_string(&(mergers[i][0]));
                read_quoted_string(&(mergers[i][1]));
        }
}

read_quoted_string(char *s)
{
        int i=0;                /* counter */
        char c;                 /* latest character */

        while ((c=getchar()) != '\"') ;
        while ((c=getchar()) != '\"') {
                s[i] = c;
                i = i+1;
        }
        s[i] = '\0';
}
```

String Matching

```
/*      Return the position of the first occurrence of the 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);
}
```

Name Replacement

```
/*      Replace the substring of length xlen starting at position pos in
        string s with the contents of string y.
*/

replace_x_with_y(char *s, int pos, int xlen, char *y)
{
        int i;                          /* counter */
        int slen,ylen;                  /* lengths of relevant strings */

        slen = strlen(s);
        ylen = strlen(y);

        if (xlen >= ylen)
                for (i=(pos+xlen); i<=slen; i++) s[i+(ylen-xlen)] = s[i];
        else
                for (i=slen; i>=(pos+xlen); i--) s[i+(ylen-xlen)] = s[i];

        for (i=0; i<ylen; i++) s[pos+i] = y[i];
}
```

Main Routine

```
main()
{
        string s;               /* input string */
        char c;                 /* input character */
        int nlines;             /* number of lines in text */
        int i,j;                /* counters */
        int pos;                /* position of pattern in string */

        read_changes();
        scanf("%d\n",&nlines);
        for (i=1; i<=nlines; i=i+1) {         /* read text line */
                j=0;
                while ((c=getchar()) != '\n') {
                        s[j] = c;
                        j = j+1;
                }
                s[j] = '\0';

                for (j=0; j<nmergers; j=j+1)
                        while ((pos=findmatch(mergers[j][0],s)) != -1) {
                           replace_x_with_y(s, pos,
                                strlen(mergers[j][0]), mergers[j][1]);
                        }

                printf("%s\n",s);
        }
}
```

Example Review

- **Error and exception handling**
- **Efficiency**
  - Space complexity
  - Time complexity

Crypt Kicker II

A popular but insecure method of encrypting text is to permute the letters of the alphabet. That is, in the text, each letter of the alphabet is consistently replaced by some other letter. To ensure that the encryption is reversible, no two letters are replaced by the same letter.

A powerful method of cryptanalysis is the known plain text attack. In a known plain text attack, the cryptanalyst manages to have a known phrase or sentence encrypted by the enemy, and by observing the encrypted text then deduces the method of encoding.

Your task is to decrypt several encrypted lines of text, assuming that each line uses the same set of replacements, and that one of the lines of input is the encrypted form of the plain text `the quick brown fox jumps over the lazy dog`.

Crypt Kicker II

Input

The input begins with a single positive integer on a line by itself indicating the number of test cases, followed by a blank line. There will also be a blank line between each two consecutive cases.

Each case consists of several lines of input, encrypted as described above. The encrypted lines contain only lowercase letters and spaces and do not exceed 80 characters in length. There are at most 100 input lines.

Output

For each test case, decrypt each line and print it to standard output. If there is more than one possible decryption, any one will do. If decryption is impossible, output

```
No solution.
```

The output of each two consecutive cases must be separated by a blank line.

Crypt Kicker II

Sample Input

```
1

vtz ud xnm xugm itr pyy jttk gmv xt otgm xt xnm puk ti xnm fprxq
xnm ceuob lrtzv ita hegfd tsmr xnm ypwq ktj
frtjrpgguvj otvxmdxd prm iev prmvx xnmq
```

Sample Output

```
now is the time for all good men to come to the aid of the party
the quick brown fox jumps over the lazy dog
programming contests are fun arent they
```

Doublets

A *doublet* is a pair of words that differ in exactly one letter; for example, “booster” and “rooster” or “rooster” and “roaster” or “roaster” and “roasted”.

You are given a dictionary of up to 25,143 lowercase words, not exceeding 16 letters each. You are then given a number of pairs of words. For each pair of words, find the shortest sequence of words that begins with the first word and ends with the second, such that each pair of adjacent words is a doublet. For example, if you were given the input pair “booster” and “roasted”, a possible solution would be (“booster,” “rooster,” “roaster,” “roasted”), provided that these words are all in the dictionary.

Input

The input file contains the dictionary followed by a number of word pairs. The dictionary consists of a number of words, one per line, and is terminated by an empty line. The pairs of input words follow; each pair of words occurs on a line separated by a space.

Output

For each input pair, print a set of lines starting with the first word and ending with the last. Each pair of adjacent lines must be a doublet.

If there are several minimal solutions, any one will do. If there is no solution, print a line: “`No solution.`” Leave a blank line between cases.

Doublets

Sample Input

```
booster
rooster
roaster
coasted
roasted
coastal
postal

booster roasted
coastal postal
```

Sample Output

```
booster
rooster
roaster
roasted

No solution.
```