Chars

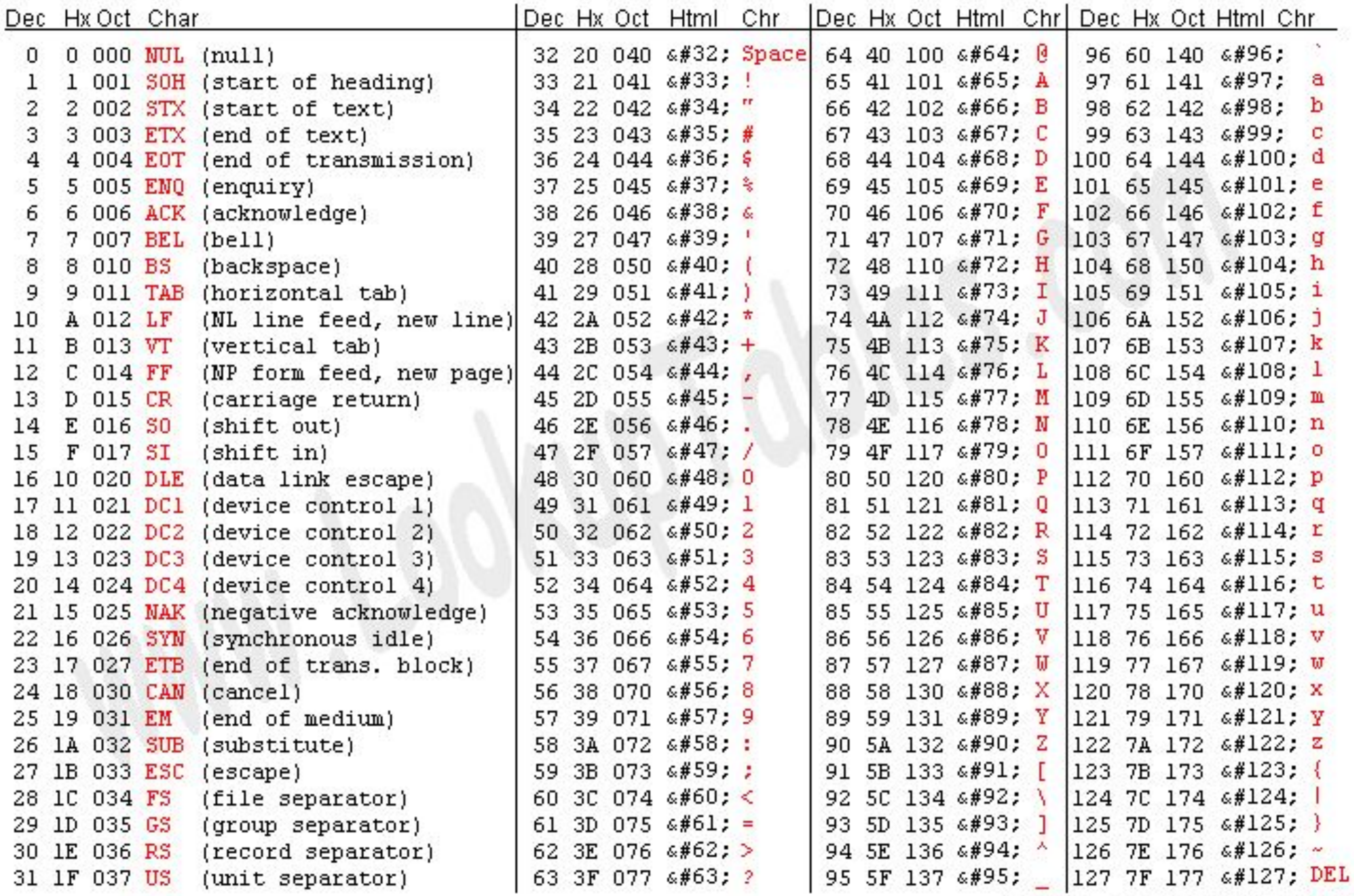
Characters

- Example 6.1

Characters are complicated

- In the old days, there was a very simple character set, ASCII, which represented the basic English language characters
- Essentially what the standard `char` type represents
- Indicate with single quotes

```
char my_char = ‘a’;
```



Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

The world is not just English

- A `char` is only 8 bits (1 byte) so it can only represent 256 characters
- Not enough to deal with the world's character sets
- Unicode is a way to represent these character sets, but it is complicated

utf8

- After a long history, a committee created a Unicode standard called utf8
    - ASCII stuff unchanged
    - Variable size byte values to store an essentially infinite number of characters

New char types

- C++ allows for new char types
  - `wchar_t`: older, implementation dependent
  - `char16_t` and `char32_t`: C++11 for unicode

Which of the following should be on the exam?

- `char`
- `wchar_t`
- `char16_t`
- `char32_t`

We’ll worry about this later

- This is just a complicated topic and we’ll not worry about it here
    - Plenty of other problems in C++

Character Operations

- Example 6.2

Character Functions

- Page 92 of the book
- These are all tests of various kinds you can place on a character
  - Most are Booleans

#include<cctype>

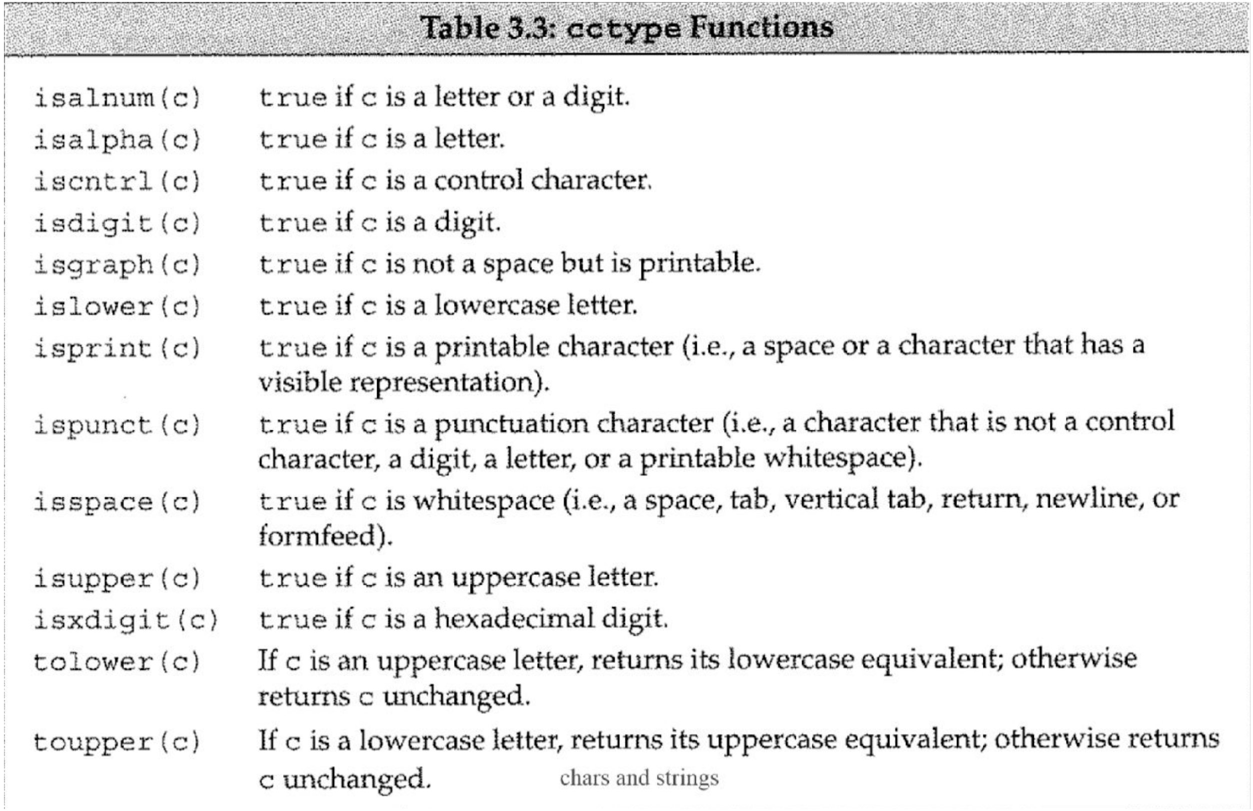


Table 3.3: cctype Functions

`isalnum(c)`	`true` if `c` is a letter or a digit.
`isalpha(c)`	`true` if `c` is a letter.
`iscntrl(c)`	`true` if `c` is a control character.
`isdigit(c)`	`true` if `c` is a digit.
`isgraph(c)`	`true` if `c` is not a space but is printable.
`islower(c)`	`true` if `c` is a lowercase letter.
`isprint(c)`	`true` if `c` is a printable character (i.e., a space or a character that has a visible representation).
`ispunct(c)`	`true` if `c` is a punctuation character (i.e., a character that is not a control character, a digit, a letter, or a printable whitespace).
`isspace(c)`	`true` if `c` is whitespace (i.e., a space, tab, vertical tab, return, newline, or formfeed).
`isupper(c)`	`true` if `c` is an uppercase letter.
`isxdigit(c)`	`true` if `c` is a hexadecimal digit.
`tolower(c)`	If `c` is an uppercase letter, returns its lowercase equivalent; otherwise returns `c` unchanged.
`toupper(c)`	If `c` is a lowercase letter, returns its uppercase equivalent; otherwise returns `c` unchanged.

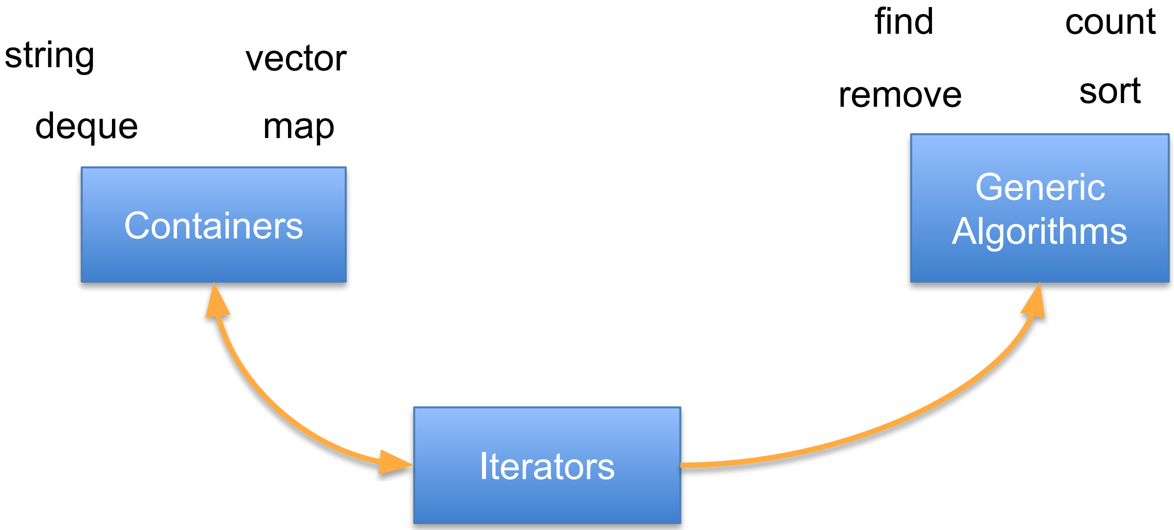
chars and strings

Strings

strings

- our first STL container

Standard Template Library (STL)



More STL

- Containers
  - Data structures to hold other data with various capabilities/efficiencies
    - Most are templated
- Generic Algorithms
  - Algorithms for common tasks that work with container contents (mostly)
- Iterators
  - A kind of pointer, allowing access to containers independent of type

String Class Library

- A string is an STL class used to represent a **sequence** of **characters**
    - An STL sequence, but not templated as it can only hold characters
    - Templated containers can hold any type
- As with other classes we have seen, there is a representation for the string objects and a set of operations
- Use `#include<string>`

What library is std::string provided in?

- `#include <string>`
- `STL`
- `The Standard Library`
- `I don't know`

Objects and Methods

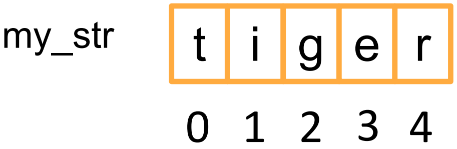
- A string is a C++ object. The word object has special meaning in programming, but there are two we care about for the moment.
  - What data it stores
  - What methods we can call

First Strings

- Example 6.3

Declaring Strings

- `string my_str;`
  - Creates a string object and initializes it to the empty string “”
- `const string my_str = “tiger”;`
  - Creates a string object with 5 characters



my_str

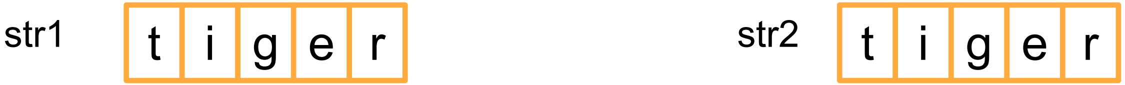
t	i	g	e	r
0	1	2	3	4

Internal Structure

- Each element in a `string` is a single character
  - `char my_char = ‘a’;`
- In this case, a string is a sequence of `char` type elements
- Thus a variable of type `string` can hold a large number of individual characters

Copy Assignment

- Declaration
  - `string str1, str2 = “tiger”;`
- Assignment
  - `str1 = str2;`
- Makes a copy of str2 so



Other ways to initialize a string

- `{}` contains universal initializer, a list of elements to go in the string
- Since strings hold characters, we list individual characters

```
string first{‘H’, ‘o’, ‘m’, ‘e’, ‘r’};
cout << first << endl;
// prints Homer
```

More initializers

- Can create copies of an individual character in a string
  - First arg is the count
  - Second arg is the characters

```
string a_5(5, ‘a’);
cout << a_5 << endl;
// prints aaaaa
```

More initializers

- Copy construction is technically different from assignment, but it does the same kind of thing

```
string first = “Homer”;
string second = first;
cout << second << endl;
```

prints `Homer`

It’s a *copy* of the original

We worry about copying

- If we copy a long string (say a copy of Shakespeare as a string) we do a lot of work
  - We have to make memory (which the string class does) to hold it
  - We have to use the CPU to move all that data around
- We will discuss this more

Methods, like functions

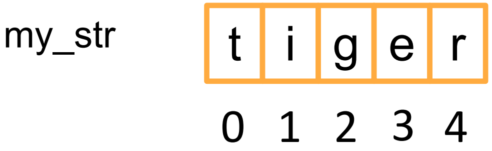
- A method is a function that is:
  - called in the context of a particular instance of an object
  - uses the dot notation for the call

Example methods size() and length()

- `string my_str = “tiger”;`
- `size()` method returns the number of characters in the string
- `cout << my_str.size();`
- Will output the integer 5
- `.length()` is the same as `.size()`

Data members and Subscripts

- To access individual characters in a string, use the `.at` member function
  - Index starts at 0
- `string my_str = “tiger”;`



my_str

t	i	g	e	r
0	1	2	3	4

- `cout << my_str.at(2);`
- outputs the character 'g'

[] instead of .at

- You can also use the subscript operator `[]`.

```
string my_string;
my_string = “hello”;
cout << my_string[4] // output is ‘o’
```

[] vs .at

- There is one important difference:
- If you access a non-existent index
  - `.at` will throw an error
  - `[]` will not (it will do something weird, but not throw an error)

Starting at 0

- On of the most important things to remember about strings (or any sequence in C++) is that they start at 0
  - Same as in Python and Java
- You will save yourself grievous headaches if you remember this!

Can assign values

- You can assign using the .at or [] operator

```
string my_str;
my_str = “hello”;
my_str[0] = ‘j’;
// string is now jello
my_str.at(0) = ‘h’;
// back to hello
```

Subscript Assignment

```
string my_str = “tiger”;
my_str.at(2) = ‘m’;
cout << my_str;
```

- Outputs “timer”

Assign Method

- You can also use the `assign` method and get *substring* assignment

```
string a_str;
a_str = “myTry”;
string next_str;
next_str.assign(a_str, 2, string::npos);
// next_str becomes “Try”
```

More String Methods

string::npos

- The `::` is the scope resolution operator
- It gives you access to functions and variables that are defined as part of a class
- `string::npos` is the name of a variable within the string class
- It stands for “no position”, a position not found in the string

Character Processing

```
string my_str = “tiger”;
for (int i = 0; i < my_str.size(); i++) {
   cout << i << “: “ << my_str[i] << endl;
}
```

Output:

```
0: t
1: i
2: g
3: e
4: r
```

not int, string::size_type

- Every STL container has a size_type.
- For strings it is string::size_type.
- You shouldn’t use use int

Whatever size returns is size_type

```
string my_str = “tiger”;
for (decltype(my_str.size()) i = 0; i < my_str.size(); i++) {
    cout << i << “: “ << my_str[i] << endl;
}
or ... string::size_type i = 0; ...
```

size_types are unsigned

- As for all unsigned types, you can get some strange behavior if you go below 0.
- Watch for that (try it, see what it prints).

Some regular functions: I/O

- Input operator `>>` is overloaded:

```
string my_str;
cin >> mystr;
```

- Reads first word in istream up to whitespace
- If input is “`fred`”, `my_str` is “`fred`”
- If input is “`mary jones`”, `my_str` is only “`mary`”

More I/O, full line input

- To read a whole line of text (up to a newline character, `‘\n’`) use
- `getline(cin, my_str);`
- If input is `“Mary Jones likes cats\n”` then `my_str` is `“Mary Jones likes cats”`
  - `‘\n’` not included (is discarded)

my_str	M	a	r	y		J	o	n	e	s		l	i	k	e	s		c	a	t	s

String input

- Example 6.4

for-each loop

- Example 6.5

for-each loop (range-based for loop)

- Much better loop
  - Similar to the for-loop in Python
  - Is a C++11 thing

```
string my_str = “tiger”;
for (auto chr : my_str)
    cout << chr << “, “;
```

C++ can determine the type of each element so we just `auto` the type

String Comparison

- Beginning at character 0 (leftmost), compare each character until a difference is found
- The ASCII values of those different characters determines the comparison value
- E.g. “aardvark” < “ant” since the second characters ‘a’ < ‘n’ because
  97 < 110

String Ops

- Example 6.6

Concatenation

- Concatenation appends one string to another.

```
string result;
string tig = “tiger”;
string ant = “ant”;
result  = tig + ant;
cout << result;
```

- Output is `“tigerant”`

Substrings

- The method is substr

```
string my_str = “abc123”;
mystr.substr(0, 4) // Starts at 0, length 4
```

  - “abc1”

- If length is past end or no length argument, assume to the end

```
my_str.substr(1, 100)
my_str.substr(1);
my_str.substr(1, string::npos)
```

Same thing

  - “bc123”

Another Initializer

- You can do this at the initializer stage

```
string last = “Simpson”;
string sub_last(last, 3, 2);
```

- copy from last
  - start at index 3
  - length of 2
  - Prints `ps`

Constructors

- Methods / functions called in the context of initializing a newly declared variable are called constructors
- Can have multiple based on arguments
- All the initializers we’ve seen are constructors
- We will write our own for our new classes later

Some general seq ops

```
string my_str = “abc”;
// push_back: append 1 element to end
my_str.push_back(‘d’); // “abcd”
// append string at end
my_str.insert(my_str.size(), “efgh”);
```

More String operations

- Table 9.13 on page 363
- www.cplusplus.com/reference/string/string

String find function

- Example 6.7

find function

- `find` finds the first occurrence of char in a string, starting at the start position.

```
string my_str = “hello world”
string::size_type pos = 0;
pos = my_str.find(‘e’, pos);
// pos gets set to 1
// doesn’t exist? return string::npos
```

Lots of find functions

- Look at Table 9.14 (page 365). Works for characters and strings
  - `s.rfind(arg)`: finds the last of arg in s
  - `s.find_first_of(arg)`: first of any of the args in s
  - `s.find_last_of(arg)`: find last of any of the args in s
  - `s.find_first_not_of(args)`: find first of any char in s that is not in arg
  - `s.find_last_not_of(args)`: find last of any char in s that is not in arg

Lychrel Number

- Example 6.8