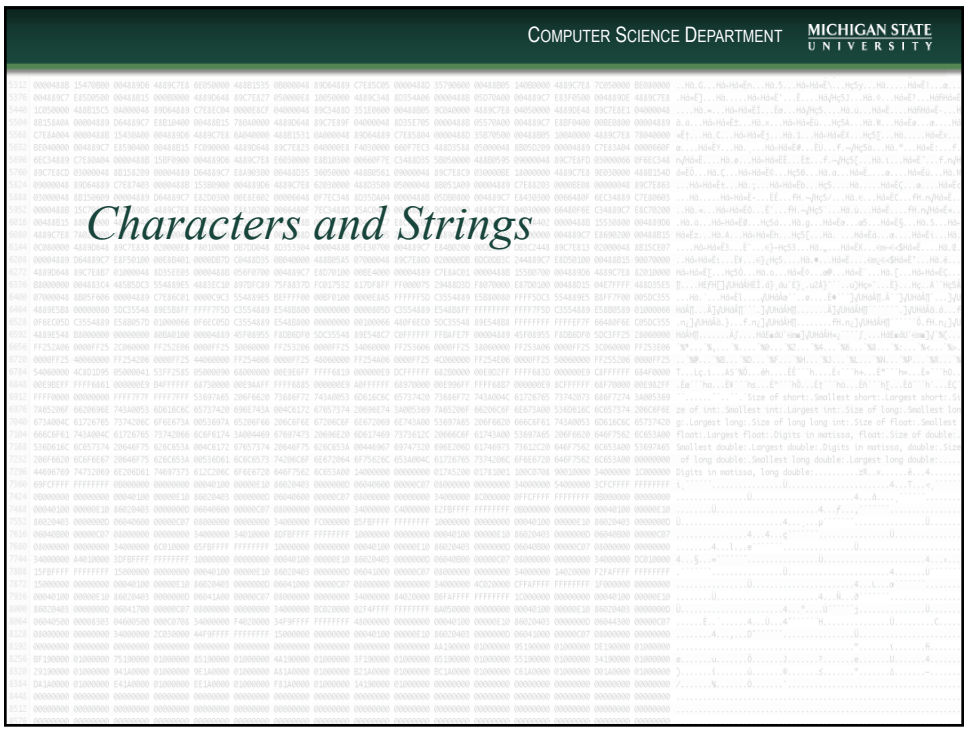
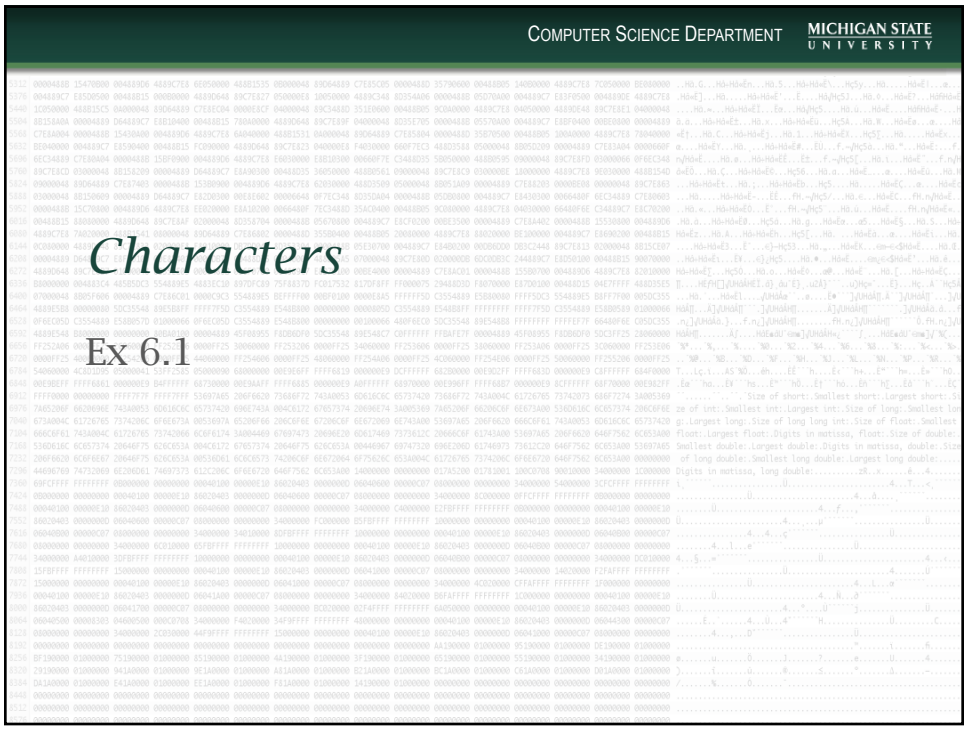

COMPUTER SCIENCE DEPARTMENT
MICHIGAN STATE UNIVERSITY
Characters and Strings


COMPUTER SCIENCE DEPARTMENT
MICHIGAN STATE UNIVERSITY
Characters
Ex 6.1

characters are complicated

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

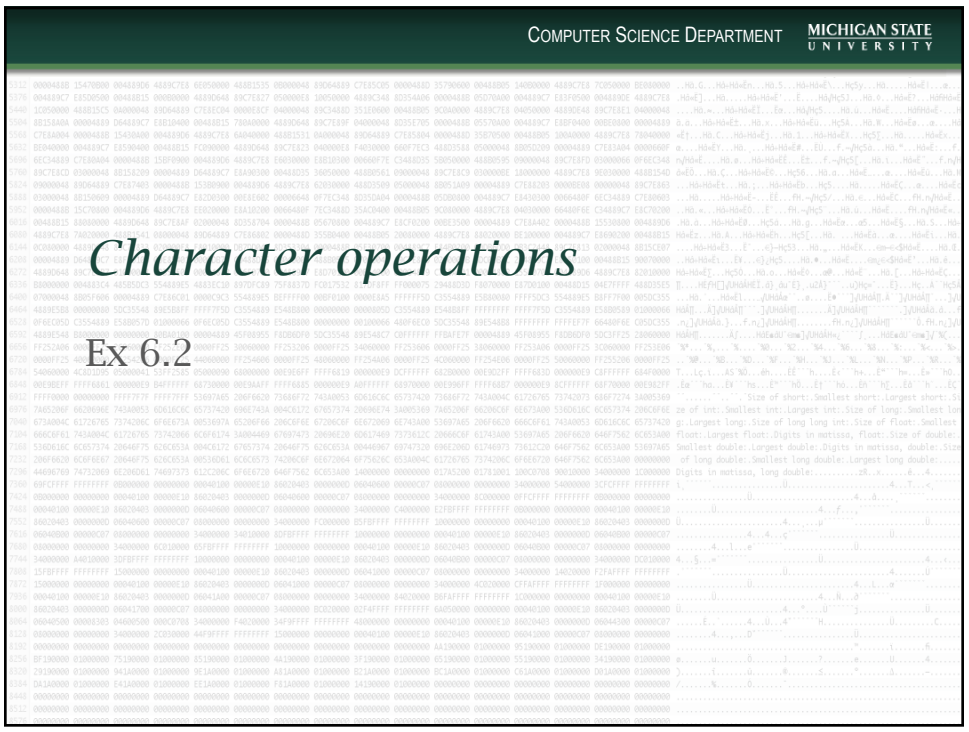
c++ allows for new char types:

- `wchar_t` : older, implementation dependent
- `char16_t` and `char32_t` : c++11 for unicode

We'll worry about this later

This is just a complicated topic and we'll worry about it later

- plenty of other problems in C++



Character operations

Ex 6.2

character functions

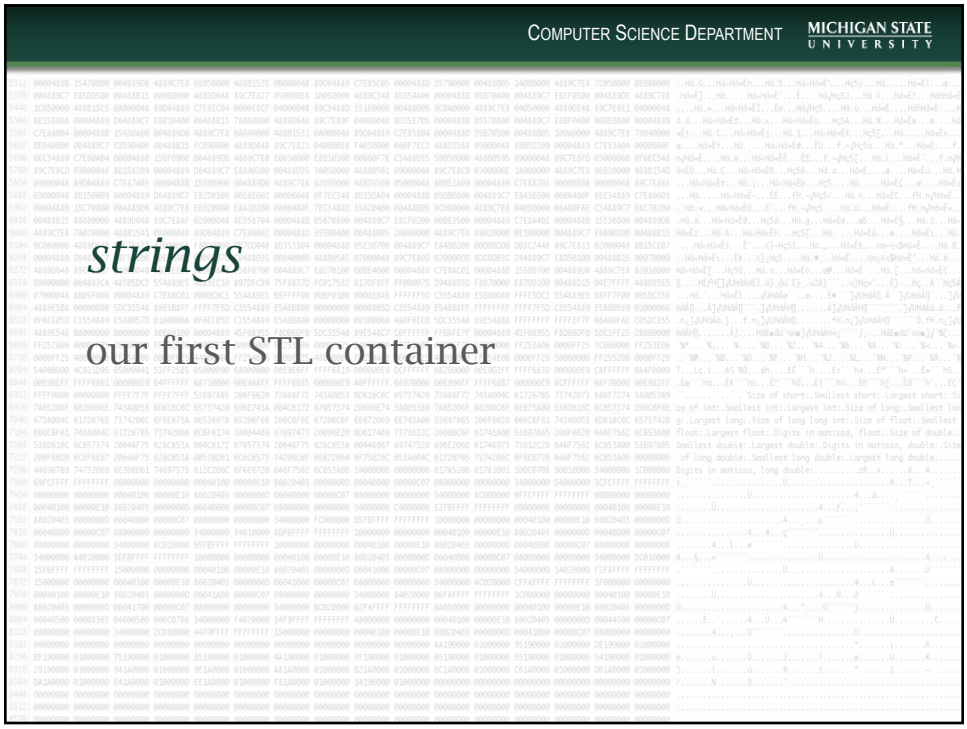
page 92 of the book

these are all tests of various kinds you can place on a character. Most are booleans.

#include<cctype>

Table 3.3: `cctype` Functions

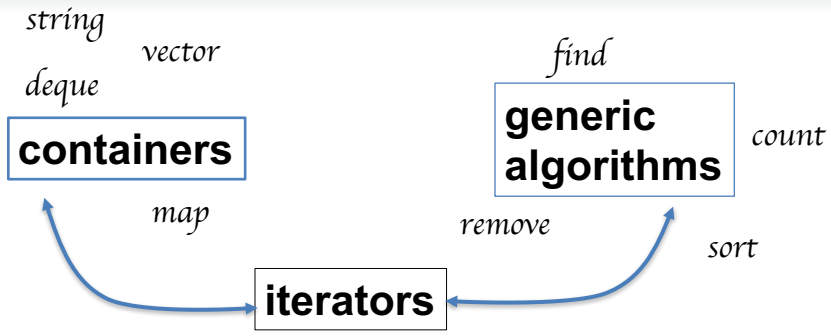
`isalnum(c)`	`true` if `c` is a letter or a digit.
`isalpha(c)`	`true` if `c` is a letter.
`iscntrl(c)`	`true` if `c` is a control character.
`isdigit(c)`	`true` if `c` is a digit.
`isgraph(c)`	`true` if `c` is not a space but is printable.
`islower(c)`	`true` if `c` is a lowercase letter.
`isprint(c)`	`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.



strings

our first STL container

Standard Template Library (STL) 3 parts



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>`

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

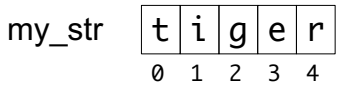
Declaring Strings

```
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 "tiger"



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

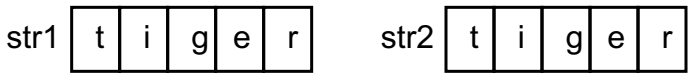
Declaration

```
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 character

```
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 new_first = first;
cout << new_first << 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();
```

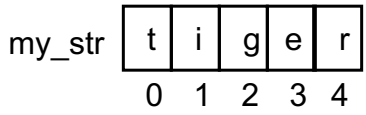
Will output the integer 5

`.length()` is the same as `.size()`

Data member:Subscript

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

```
string my_str = "tiger";
```



- `cout << my_str.at(2);`
- outputs `'g'` (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 an non-existant index, `.at` will throw an error, `[]` will not (it will do something weird, but not throw an error)

Starting at 0

One of the most important things to remember about strings (any sequence of things in C++) is that they start at 0.

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` operator 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"
```


string::npos

The `::` is the scope resolution operator. It gives you access to functions and variables that are defined as part of a class. So `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

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

Every STL container has a `size_type`. For strings it is `string::size_type`. Though you can get away with `int`, you should not. Instead:

whatever size returns a size_type

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

size_types are unsigned

As with all unsigned types, you can get some strange behavior if you go below 0.

Watch for that (try it, see what it prints).

string input

Example 6.4

Some regular functions: I/O

- Input operator >> is overloaded:

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

- 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"`, 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	...

range based for loop

Example 6.5

range based for

Much better, range based for loop

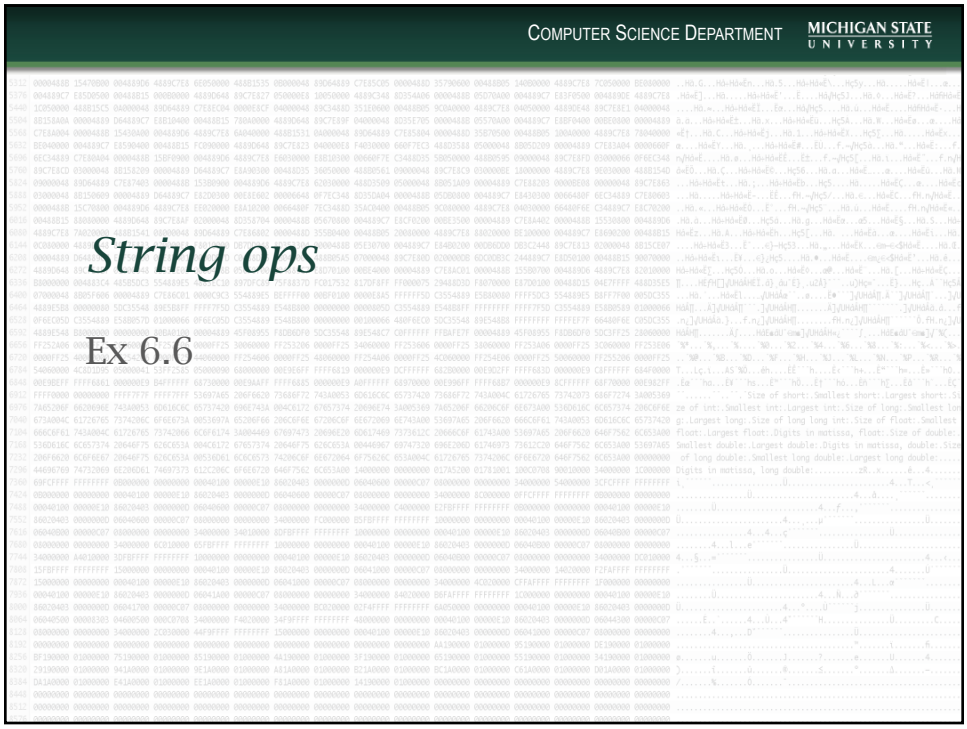
- this is the for loop in Python!
- it's a c++11 thing

```
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



Concatenation

Concatenation appends one string to another.

```
string result;
string tig = "tiger"
string ant = "ant";
result = tig + ant;
cout<< result;
```

- Output is `"tigerant"`

Ex 6.6, substrings

The method is substr

```
string my_str = "abc123";
my_str.substr(0,4)   //start at 0, len 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");
```


Table 9.13: Operations to Modify strings

`s.insert(pos, args)`	Insert characters specified by *args* before *pos*. *pos* can be an index or an iterator. Versions taking an index return a reference to `s`; those taking an iterator return an iterator denoting the first inserted character.
`s.erase(pos, len)`	Remove `len` characters starting at position `pos`. If `len` is omitted, removes characters from `pos` to the end of the `s`. Returns a reference to `s`.
`s.assign(args)`	Replace characters in `s` according to *args*. Returns a reference to `s`.
`s.append(args)`	Append *args* to `s`. Returns a reference to `s`.
`s.replace(range, args)`	Remove *range* of characters from `s` and replace them with the characters formed by *args*. *range* is either an index and a length or a pair of iterators into `s`. Returns a reference to `s`.

***args* can be one of the following; `append` and `assign` can use all forms**
`str` must be distinct from `s` and the iterators `b` and `e` may not refer to `s`

`str`	The `string str`.
`str, pos, len`	Up to `len` characters from `str` starting at `pos`.
`cp, len`	Up to `len` characters from the character array pointed to by `cp`.
`cp`	Null-terminated array pointed to by pointer `cp`.
`n, c`	`n` copies of character `c`.
`b, e`	Characters in the range formed by iterators `b` and `e`.
initializer list	Comma-separated list of characters enclosed in braces.

***args* for `replace` and `insert` depend on how *range* or *pos* is specified.**

`replace (pos,len,args)`	`replace (b,e,args)`	`insert (pos,args)`	`insert (iter,args)`	*args* can be
yes	yes	yes	no	`str`
yes	no	yes	no	`str, pos, len`
yes	yes	yes	no	`cp, len`
yes	yes	no	no	`cp`
yes	yes	yes	yes	`n, c`
no	yes	no	yes	`b2, e2`
no	yes	no	yes	*initializer list*

string find function

Example 6.7

find function

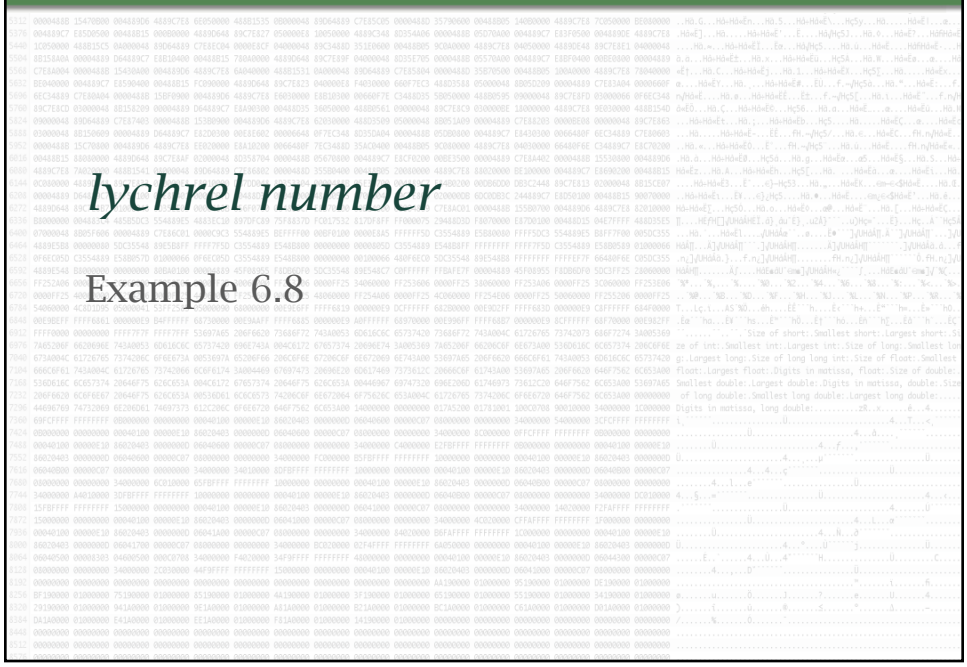
find function finds the first occurrence of char in a string, starting at the start position.

```
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 (pg 365). Works for characters and strings

- s.rfind(arg): find 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