

today: strings

- what are strings and why to use them
- reading: textbook chapter 8

what are strings

- a *string* in C++ is one of a special kind of data type called a *class*
- we will talk more about *classes* in detail at the end of the term
- but note that we have already used two classes when we covered files: the *ifstream* class and the *ofstream* class
- a class is a compound data type, unlike the simple, native data types we've already discussed (e.g., *int*, *char*, *bool*, *double*, etc)
- a class has *members*:
 - it has *data* fields and *functions*

strings: declaring and initializing

- strings are declared like this:

```
string s
```

where *s* is a variable whose data type is a *string*
- you can set the value of the string using the assignment operator and double quotes ("`s = "hello";`");
- NOTE that you use *single* quotes for *char* values and *double* quotes for *string* values:

```
char c = 'A';  
string s = "hello";
```
- ALSO NOTE that when you use the *string* class, you also need to include the *string* header file:

```
#include <string>  
using namespace std;
```

strings: output

- we have already used strings for output, e.g.:

```
cout << "hello" << endl;
```
- but we have not yet output variables that were declared as strings
- but we can... it's just like this:

```
#include <iostream>  
#include <string>  
using namespace std;  
  
int main() {  
    string s = "hello";  
    cout << s << endl;  
} // end of main()
```

strings: input

- there are two ways to read input values from the keyboard (or from a file) into a string variable:
 - (1) using the >> operator
 - (2) using the `getline()` function
- the first way, using the >> operator, will only read until the first *whitespace* character is read (the term “whitespace” refers to characters like blank spaces, tabs and newlines)
- when reading a string variable using the >> operator, the input will stop as soon as the first whitespace character is read
- example (on the next page):

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string s;
    cout << "please enter your name:";
    cin >> s;
    cout << "s = " << s << endl;
} // end of main()
```

- if the user enters *david ortiz* when the program asks
please enter your name:
then the value of `s` will be “david”

- HOWEVER, when reading a string variable using the `getline()` function, the input will stop as soon as the first *newline* character is read (i.e., the user hits the ENTER key on the keyboard), e.g.:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string s;
    cout << "please enter your name:";
    getline( cin, s );
    cout << "s = " << s << endl;
} // end of main()
```

- here, if the user enters *david ortiz* when the program asks
please enter your name:
then the value of `s` will be “david ortiz”

- note that the `getline()` function can optionally take a third argument, a *delimiter*
- this would be for cases where you wanted to stop the input not at whitespace or at a newline, but at some other character— called the *delimiter*
- suppose you wanted to enter several commands for a robot to follow using one string, like this:
forward|wait|backward|turn left|stop
- you could do this using the | (vertical bar) character as the delimiter and the `getline()` function with three arguments, e.g.:

```
string s;
cout << "enter some commands: ";
getline( cin, s, '|' );
```

If the user entered `forward|backward` when prompted by the above program, the value of `s` would be `forward`, since the input would stop at the delimiter (`|`). You would have to call `getline(cin, s, '|');` again to read the next command (e.g., `wait`).

strings: operators

- there are several operators that work with strings
- the plus sign (+) is the *concatenation* operator, e.g.:

```
string s1, s2, s3;  
s1 = "david ";  
s2 = "ortiz";  
s3 = s1 + s2;
```

After the above code fragment, the value of s3 will be "david ortiz"

- the *comparison* operators also work with strings
(==, <, <=, >, >=)

- the double equals sign (==) compares the value of two strings and returns true if they are the same, e.g.:

```
string s1, s2, s3;  
bool a1, a2;  
s1 = "david ";  
s2 = "ortiz";  
s3 = "david ";  
a1 = ( s1 == s2 );  
a2 = ( s1 == s3 );
```

After the above code fragment:
the value of a1 will be false
and
the value of a2 will be true

- the inequality operators (<, <=, >, >=) perform a *lexical comparison* between two strings
- a "lexical comparison" is like checking if two strings are in alphabetical order: one is less than the other if it comes before the other alphabetically
- EXCEPT, the lexical comparison is *case sensitive* and uses the ASCII table, which means that all the upper case letters (A..Z) come before (are less than) all the lower case letters (a..z), e.g.:

```
string s1, s2, s3;  
bool a1, a2;  
s1 = "ABC";  
s2 = "DEF";  
s3 = "abc ";  
a1 = ( s1 < s2 );  
a2 = ( s3 < s2 );
```

After the above code fragment:
the value of a1 will be true because "ABC" < "DEF"
and
the value of a2 will be false because "abc" > "DEF"

strings: indexes

- a string is like an array of char
- so you can use the *index* of the individual characters of the string just like you can use the indexes of the individual elements of an array, like the arrays of ints you created for the last homework assignment
- if you have:
string s = "ortiz";
then: s[0] is assigned the value o (the letter "oh")
s[1] is assigned the value r
s[2] is assigned the value t
s[3] is assigned the value i
s[4] is assigned the value z
- you can also use the member function at() to find the value of an individual character of a string
e.g., instead of using s[3], you can use s.at(3)

strings: length

- if you have:
`string s = "ortiz";`
then the *length* of the string is 5
- there are two *member functions* of the string class that will tell you the length of a string: `length()` and `size()` (they do the same thing)
you call them like this:

```
string s1;  
int n1, n2;  
s1 = "ortiz";  
n1 = s1.length();  
n2 = s1.size();
```

After this code fragment,
the value of `n1` will be 5
and so will the value of `n2`

strings: searching

- the `find()` member function is used to locate a substring within a primary string
- the function returns the value of the index in the primary string at which the substring starts, if the substring exists in the primary string;
or else the function returns the constant `string::npos`
- for example:

```
string s1 = "david ortiz";  
int n1, n2;  
n1 = s1.find( "avid", 0 );  
n2 = s1.find( "ask", 0 );
```

After the above code fragment:
the value of `n1` will be 1
the value of `n2` will be `string::npos`
- the first argument to the `find()` function is the substring to search for
- the second argument to the `find()` function is the index in the primary string at which to start searching; 0 means to start searching at the beginning of the primary string

strings: editing

- there are three *editing* member functions that are part of the string class:
 - `insert()`
 - `replace()`
 - `erase()`
- the `insert()` function inserts a substring into the primary string
- the `replace()` function replaces a substring with another substring within the primary string
- the `erase()` function erases a number of characters within the primary string
- example (on the next page):

```
#include <iostream>  
#include <string>  
using namespace std;  
int main() {  
    string s = "ortiz";  
    cout << "first, s=" << s << endl;  
    s.insert( 0, "david " );  
    cout << "second, s=" << s << endl;  
    s.replace( 0, 1, "D" );  
    s.replace( 6, 1, "0" );  
    cout << "third, s=" << s << endl;  
    s.erase( 1, 4 );  
    cout << "fourth, s=" << s << endl;  
} // end main()
```

The output of the above program will be:

```
first, s=ortiz  
second, s=david ortiz  
third, s=David Ortiz  
fourth, s=D Ortiz
```

strings: parsing

- the `substr()` member function is used to extract a substring from within a primary string
- example:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string s1 = "D Ortiz";
    string s2;
    cout << "s1=" << s1 << endl;
    s2 = s1.substr( 2, 5 );
    cout << "s2=" << s2 << endl;
} // end main()
```

The output of the above program will be:

```
s1=D Ortiz
s2=Ortiz
```