String class

TA: You Lu
A string literal is just some text in quotes eg:

“Homer Simpson”

We have already used string literals in our code with cout

cout << “Homer Simpson”;
string variables

We would like to have variables that store string literals.

First we need to include the string library:

```
#include <string>
```

We place this line just after the line `#include <iostream>` (or just before).

Now in our code we can declare string variables!

```
string name = "Homer Simpson\n";
cout << name;
```

Note the string text can hold punctuation, spaces, and even escape characters.
In many ways string variables behave exactly like the int or double variables:

```cpp
string first_name;
string last_name;

cout << "What is your first name? ";
cin >> first_name;
cout << "What is your last name? ";
cin >> last_name;
```

We can even have something like:

```cpp
string first_name;
string last_name;

cout << "What is your name? ";
cin >> first_name >> last_name;
```
Careful:

string boss;

// Say input is M. Burns

// output M.

// cin will only take in one word. Word is determined by white space.

To get a whole line untill a newline character use the getline command.

getline(cin, boss);  // Reads M. Burns and stores it in boss.
Concatenation is appending one string to the end of another.

We use + to concatenate two strings.

```cpp
string first_name, last_name, full_name;

cout << "What is your first name?";
cin >> first_name;
cout << "What is your last name?";
cin >> last_name;

cstring = first_name + last_name;

/* This is almost right, but there is a minor problem.  
See next slide for answer. */
When concatenating two strings together, think carefully what the result will be.

If we just concatenate the first name and the last name together there will be no space in between.

in previous example `full_name` would have the value HomerSimpson

```cpp
cout << "What is your first name?";  
cin >> first_name; // Homer  
cout << "What is your last name?";  
cin >> last_name; //Simpson

full_name = first_name + " " + last_name; // This works!
```
string boss = "M. Burns\n";

This is what the string looks like stored inside the boss variable:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>.</td>
<td>B</td>
<td>u</td>
<td>r</td>
<td>n</td>
<td>s</td>
<td>\n</td>
<td></td>
</tr>
</tbody>
</table>

Note how the first letter is stored at slot 0.

Escape characters count only as one character.
String variables are not primitive data types like int and double. They have some extra powers.

```java
string boss = "M. Burns\n";
int string_length;

string_length = boss.length(); // string_length has now value 9

Wow! This is convenient. We can get the length of a string being stored in a string variable just by writing variable_name.length()!
string first_name;
string last_name;

cout << "Please write in your full name. ";
cin >> first_name >> last_name;

cout << "Your first name has " << first_name.length() << " letters and your last name has " << last_name.length() << " letters.\n";

cout << "The total number of letters in your name is: " << first_name.length() + last_name.length();
We can extract part of a string with the substring function.

The syntax is:

```cpp
string_variable.substr (int start ,int length)
```

```cpp
string employee = "Homer Simpson";
```

```
         0 1 2 3 4 5 6 7 8 9 10 11 12
H o m e r   S i m p s o n
```

```cpp
string first_name = employee.substr(0,5);  
```

```cpp
first_name is now the string "Homer";
```
string last_name = employee.substr(6,7);

last_name is now the string “Simpson”;
By now it should be apparent that strings are not ordinary variables. They are an example of a more general type of data called classes.

Classes are soupped up data types that can not only store a value, but usually have member functions that can give information about the value or perform operations on the value.

The example you should have in your mind is the string variable. Not only does it store a value (a string), but it also has member functions length and substr.

Having data types like integers is all well and good but programmers realized they wanted to create their own more complicated data types. So they invented classes.

Eventually we will make our own classes, but for now, We will learn about classes gradually by using some premade classes.
Classes

Let's look at a simple variable declaration:

```java
int number;
```

`int` specifies the data type and `number` is the variable.

Similarly if we have a class called `My_Class`. Then line

```java
My_Class my_object;
```

declares `my_object` as an "instance" of class `My_Class`.

You can think of `My_Class` being a blue print for a building and the line

```java
My_Class my_object;
```

creates a building named `my_object` that has the structure given by the blue print `My_Class`. 

If you try the following

```cpp
cout << sqrt(2.0);
```

Depending on the computer you are at the output might be:

1.41421 or 1.4142 etc.

You cannot be sure unless you control the precision!
The way to do this is with the following magic formula

```cpp
cout << fixed;
cout << setprecision(2); // 2 sets the decimals shown to 2 places.
```

You need only to have `cout << fixed;` once in your code.

If you want to change the number of decimals shown to say 5 then write

```cpp
cout << setprecision(5);
```

later in your code.

To use these you need to include a new library

```cpp
#include <iomanip>
```
What does `setprecision(n)` do? It shows n significant digits. If you use in conjunction with fixed then you show up to n numbers after decimal point.

```cpp
double a=12345.12345;
cout << fixed;
cout << setprecision(4);
cout << a;    // output 12345.1234

// This is not the way to round!!!
```

Without fixed this only shows first 4 significant digits

For this example assume we do not have the line `cout << fixed;` in our code:

```cpp
cout << setprecision(6);
double a=12345.12345;
cout << a; //output 12345.1
Sometimes we want to format our output into neat columns. Easy way to do this is use `setw` with `cout`.

```cpp
cout << setw(10) << "One" << setw(10) << "Two\n";
cout << setw(10) << "Three" << setw(10) << "Four\n";
```

Result

```
       One       Two
      Three     Four
```

`setw(10)` says that width of 10 characters should be reserved for what follows.

The strings will be aligned to the right and if there are less than 10 characters in the string, what remains will be filled with blank spaces.
string first_name;
string last_name;

cout << "Please give me your first name ";
cin >> first_name;

cout << "Please give me your last name ";
cin >> last_name;

cout << setw(first_name.length() + 5) << first_name;
cout << setw(last_name.length() +5) << last_name;