<u>Strings and the String Class</u>

We've already seen the string data type used in some of our programs so far.

We know that it is for storing groups of characters (or words) but it is also so much more than that.

That's because the string type is not native to the C++ language and that is why you need to have:

```
#include <string>
```

at the start or programs that you are declaring a string type.

<u>Inputting Strings</u>

When inputting strings we've been able to use the usual commands:

```
string name;
cout << "What is your name? ";
cin >> name;
```

However, we run into an issue if we try and put a space in between the words such as your first and last name.

The cin object gets confused when it comes across the space character and tends to exit out the program immediately without stopping for the rest of the inputs.

To fix this we use a different technique called the **<u>getline method</u>**.

When using getline the parenthesis are used to specify the input stream object (cin) and the string object you want to input into.

```
string MyString;
cout << "What is your name? ";
getline(cin, myString);
```

Flushing the Input Stream

The cin object is often referred to as the **input stream**.

Characters are lined up in the input stream as keys are pressed on the keyboard.

Each character, however, must wait to be processed.

Therefore, if a prompt for input doesn't use every character in the input stream, the remaining characters wait in the input stream for the next prompt.

Using statements such as

```
cin >> x;
```

for inputting numbers and getline for inputting strings works well, until you try and use them together.

The problem is that after you input the number using a statement like cin, the new line character (ENTER) that you press stays in the input stream.

This isn't a problem all the time, it depends on what the next input is. Another cin to get a number won't have a problem, a string will.

To fix this problem we need to remove or flush the extra characters from the input stream.

To get rid of the extra characters use the following command after getting a number input before a string:

```
cin.ignore(80, '\n');
```

This tells the program to ignore the next 80 characters in the stream. The '\n' tells the function to stop ignoring when it gets to the newline character.

String Operations

Programs that use strings often need to perform a variety of operations on the strings that it's stored.

For example to properly center a string you may need to know how many characters long it is. You may need to add strings together.

One of the reasons that classes such as the string class are so powerful is that when we declare an string it's actually an **object** that can do more than hold data.

Objects can perform operations on the data that they hold.

Messages

One of the important concepts behind objects is that they use the idea of **containment** to hide the way it stores and manipulates data.

This keeps you from having to do all of the coding yourself or seeing all that pre-built code in your program.

To make an object do what we want it to do we send the object a **message**.

We don't care how it accomplishes this task we just care about the result of the message.

Obtaining the Length of a String

The message used to obtain the length of a string is simply *length*.

For example:

```
int l;
string MyString2;
MyString2 = "ABCDEFG";
l = MyString2.length();
```

If we look at this in pieces we see that an integer for the length (l) is declared along with a string object.

That string object is assigned some value.

The assignment operator (=) assigns the value returned by the string object to the variable *l*. In this case the string's length.

The period that follows the object name is known as the <u>**dot operator**</u>.

<u>**Concatenation**</u>

The operation of adding one string onto the end of another string is known as <u>**concatenation**</u>.

For example if you wanted to combine two strings that had first name and last name together you would have to concatenate them. (Don't forget you'd have to put in a space too!)

To accomplish this the string class tries to make this operation as easy as possible.

They do this with the use of the <u>**compound operator**</u>.

The operator is +=.

This is shorthand for adding a value to an existing value.

For example:

```
        j += 7;         is the same as          j = j + 7;

        k += n;         is the same as          k = k + n;
```

For strings this could look like:

```
MyString1 += MyString2;          'Adds MyString2 to the end of MyString1

MyString1 += "string literal";   'Add a string literal to the end of MyString1
```

```
MyString1 += Ch;                'Add a character to the end of MyString1

MyString1 += 'A';           'Add a character literal to the end of MyString1
```