STRINGS

Today

- Recap a little on arrays.
- Introduce strings.
- Show how strings and arrays are related.

Recapping arrays

- We talked before about how we define arrays and set values for the members of the array.
- For example:

  ```
  char s[5] = {'S', 'i', 'm', 'o', 'n'}
  ```

  defines an array of characters, called a, and sets its elements to have the letters of the word `Simon`.
- We can access the elements of the array using the index notation:

  ```
  a[2]
  ```

  for example, refers to the element of the array a with number 2.

- This is a character (since a is an array of characters.
- We can do to `a[2]` exactly the same things we can do to any character variable.
- For example we can print it out:

```
cout << a[2];
```

  which will print out:

```
m
```

- We can also assign it a new value:

```
a[3] = 't';
```

Strings

- To deal with strings, we need to add:

  `#include<string>`

  at the start of our program.

- With that in place, we can define variables whose type is `string`:

```
string s1 = ''Hello'';
string s2 = ''Simon'';
string s3, s4;
```

- This defines `s1` to be a string variable whose value is the word `Hello`, and `s2` to be a string variable who value is the word `Simon`.

- It also defines `s3` and `s4` to be strings, but does not give them a value.

- Since `s1`, `s2`, and `s3` are variables, we can do a lot of the kinds of things we can do to other variables to them.
- We can assign values to them and print their values out.
- For example:

```
s3 = s2;
cout << s3;
```

will generate:

```
Simon
```

- You can also test the value of two strings
- The expression

  ```
  s1 == s2
  ```

  will return `true` if the letters in the same location in both strings are the same.
- This won't be true since the first letters, `H` and `S` are different.
- However, given the value we assigned to `s3`:

  ```
  s1 == s2
  ```

  will return `true`.

- Another expression we can evaluate is:

  ```
  s1 < s2
  ```

- (We might want to use this in an `if`).
- C++ evaluates `s1 < s2` by taking the first character in `s1` and seeing if it is less than the first character in `s2`.
- If no, then it treturns `false`.
- If yes, it asks the same question of the second character in both strings.
- If every character in `s1` is less than the corresponding character in `s2`, then `s1 < s2` will be true. Otherwise it will be false.

- How does C++ tell whether one character is less than another?
- It uses the ACSII value (which we talked about in lecture I.2).
- All you probably need to know about these values is that:

```
0 < 1 < 2 < 3 < ...< 9
9 < A < B < C < ...< Z
Z < a < b < c < ...< z
```

- So `Hello` is not less than `Simon`, because o is not less than n.
- But `Hella` *is* less than `Simon`

Concatenation

- One operation that is specific to strings is *concatenation*
- For example:

```
s3 = s1 + s2;
cout << s3;
```

- The first line tells C++ to *concatenate* `s1` and `s2` and assign the result to `s3`.
- Thus `s3` now has the value of `s1` followed by the value of `s2`.

- When we print, we get:

```
HelloSimon
```

- There is no space because neither `s1` or `s2` has a space.

```
s3 = s1 + '' '' + s2;
cout << s3
```

would produce:

```
Hello Simon
```

Reading in strings

- One way to read in a string from the user is

  ```
  cin >> s3;
  ```

- This is fine if you want to read in strings like:

  ```
  Hello
  ```

  and

  ```
  Roustabout
  ```

  but no good if you want to read in:

  ```
  What time is love?
  ```

- The problem is that `cin` stops reading at the first *whitespace*.
- So, if our program has:

```
cout << ''Now type a string'';
cin >> s3;
```

and the user types:

```
What time is love?
```

in response to the prompt, then `s3` will have the value `What`.

- The way around this problem is to use the function `getline`.
- There are two ways to use `getline`.
- Like this:

```
cout << ''Now type a string'';
getline(cin,s3);
```

  it will read everything up to the point the user hits the return key, and assign this to `s3`.

- This is fine for reading in `What time is love?`

- We can also call `getline` with a third parameter.
- This parameter is a character, called a *delimiter*, which tells `getline` when to stop reading.
- If our program has:

```
cout << ''Now type a string'';
getline(cin,s3,',');
getline(cin,s4,'.');
```

and the user types:

```
First we take Manhattan, then we take Berlin.
```

then ...

- `s3` will have the value

  `First we take Manhattan`

  and `s4` will have the value

  `then we take Berlin`

- Note that the delimiters are not read in, and so don't end up in either string.

- We can also use `getline` to read strings from a file.
- For example

```
ifstream myInputFile;
ifstream.open(''sequence.txt'');
getline(myInputFile,s3);
```

will read the first line of the file `sequence.txt` into the string variable `s3`, while

```
ifstream myInputFile;
ifstream.open(''sequence.txt'');
getline(myInputFile,s4,'t');
```

will read the first line of the file `sequence.txt` up to the first `t` into the string variable `s4`.

From strings back to arrays

- As we hinted at the end of last lecture, strings are just arrays of characters.
- A string variable like `s1` is just another way of dealing with an array of characters like a that we started the lecture with.
- As a result we can do things like:

```
s2[1] = 'u';
cout << s2;
```

to produce:

```
Sumon
```

- And we can use a `for` loop to manipulate an array.
- For example:

```
for(i = 4; i >= 0; i--)
{
  cout << s2[i];
}
```

will produce:

```
nomuS
```

Summary

- This lecture started to look at strings.
- We briefly recapped arrays.
- We described how to define strings, and what operations you can carry out on them.
- We described at length how to read them in.
- Finally, we dealt briefly with the relationship between strings and arrays.