Module 30:
The string class

Intro to Computer Science 1 - C++
Professor Scott Frees

Textbook

Strings are introduced in Chapter 4

More detail on objects is found in Chapters 9 and 10

Objects

C++ is an ***object oriented*** language.

CMPS 148 is mostly about objects and classes- but we can begin discussing this concept now..

An ***object type*** is different from a *primitive* type in they have *methods* associated with them.

Object methods

- Where primitive data types can be operated on with *operators*, object types often have many *methods* that allow you to do things with them.
- For example - there is a built-in **string** data type in C++ (#include <string>)
- You may create variables of type "string"

```
string a;
string b;
```

Object methods

string variables can be assigned, using the = sign

```
string a;
string b;
```

a = “Hello”;

b = “World”

You can also compare them with
`==`, `!=`, `<`, `<=`, `>`, `>=`

And also concatenated with the + sign

```
string c = a + “ “ + b;
```

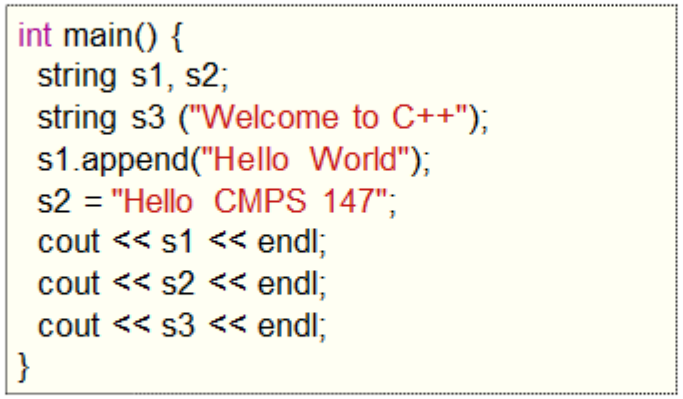
Object methods

But you can also find **their length**, find substrings, and erase segments… using *method syntax*

```
string a = “ABC”;
string b = “ABCDEFG”;
int length_a = a.length();
int length_b = b.length();
```

The variable appears to the left of the .
The function name is on the right
The function’s answer will be based on the variable it was called with.

Initialization



```
int main() {
  string s1, s2;
  string s3 ("Welcome to C++");
  s1.append("Hello  World");
  s2 = "Hello  CMPS  147";
  cout << s1 << endl;
  cout << s2 << endl;
  cout << s3 << endl;
}
```

```
> Hello World
> Hello CMPS 147
> Welcome to C++
```

You may use the = initialization style interchangeably with the *constructor* syntax

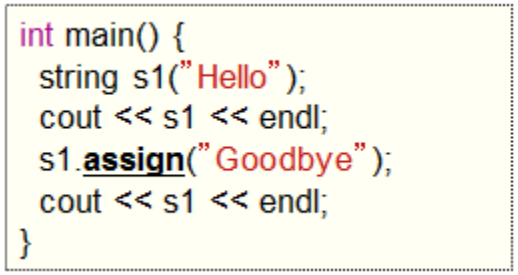
Append

```
int main() {
  string s1, s2;
  string s3 ("Welcome to C++");
  s1.append("Hello  World");
  s2 = "Hello  CMPS 147";
  cout << s1 << endl;
  cout << s2 << endl;
  cout << s3 << endl;
  s1.append(”  - great to be here!” );
  cout << s1 << endl;
}
```

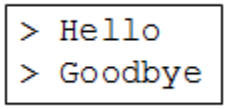
```
> Hello World
> Hello CMPS 147
> Welcome to C++
> Hello World - great to be here!
```

This is equivalent to `s1 += “ - great to be here”`

Assignment



```
int main() {
  string s1("Hello");
  cout << s1 << endl;
  s1.assign("Goodbye");
  cout << s1 << endl;
}
```



```
> Hello
> Goodbye
```

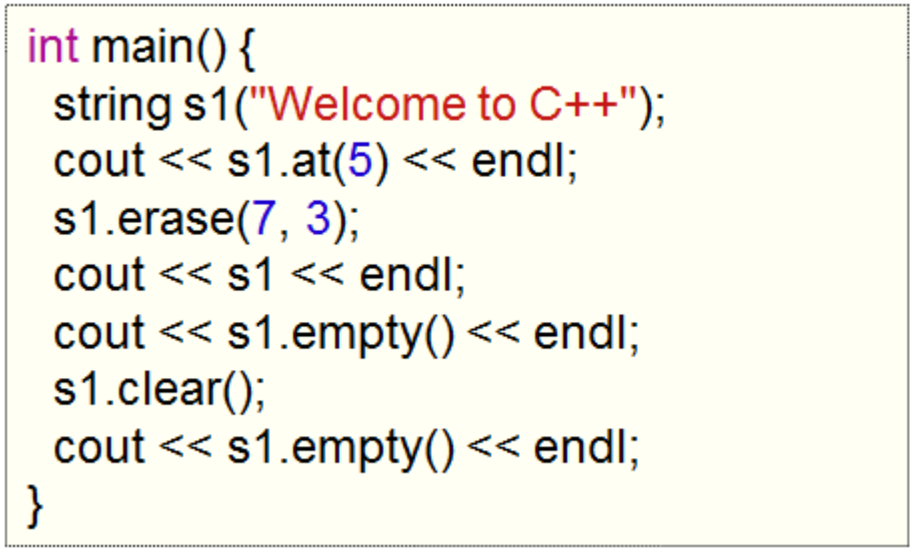
Equivalent to s1 = “Goodbye”

String manipulation

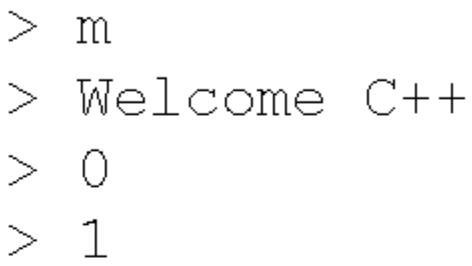
```
int main() {
  string s1("Welcome to C++");
  cout << s1.length() << endl;
  cout << s1.size() << endl;
  cout << s1.c_str() << endl;
  string s2("Welcome to C");
  if ( s1.compare(s2) != 0 )  {
     cout << "C and C++ are NOT the same!" << endl;
  }
}
```

```
> 14
> 14
> Welcome to C++
> C and C++ are not the same!
```

Erasing characters



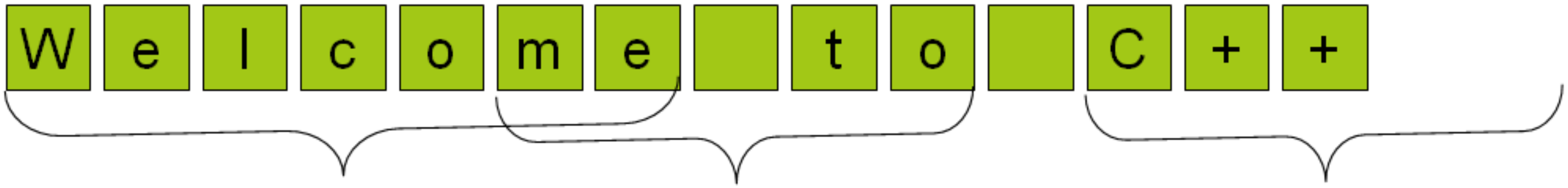
```
int main() {
  string s1("Welcome to C++");
  cout << s1.at(5) << endl;
  s1.erase(7, 3);
  cout << s1 << endl;
  cout << s1.empty() << endl;
  s1.clear();
  cout << s1.empty() << endl;
}
```



```
> m
> Welcome C++
> 0
> 1
```

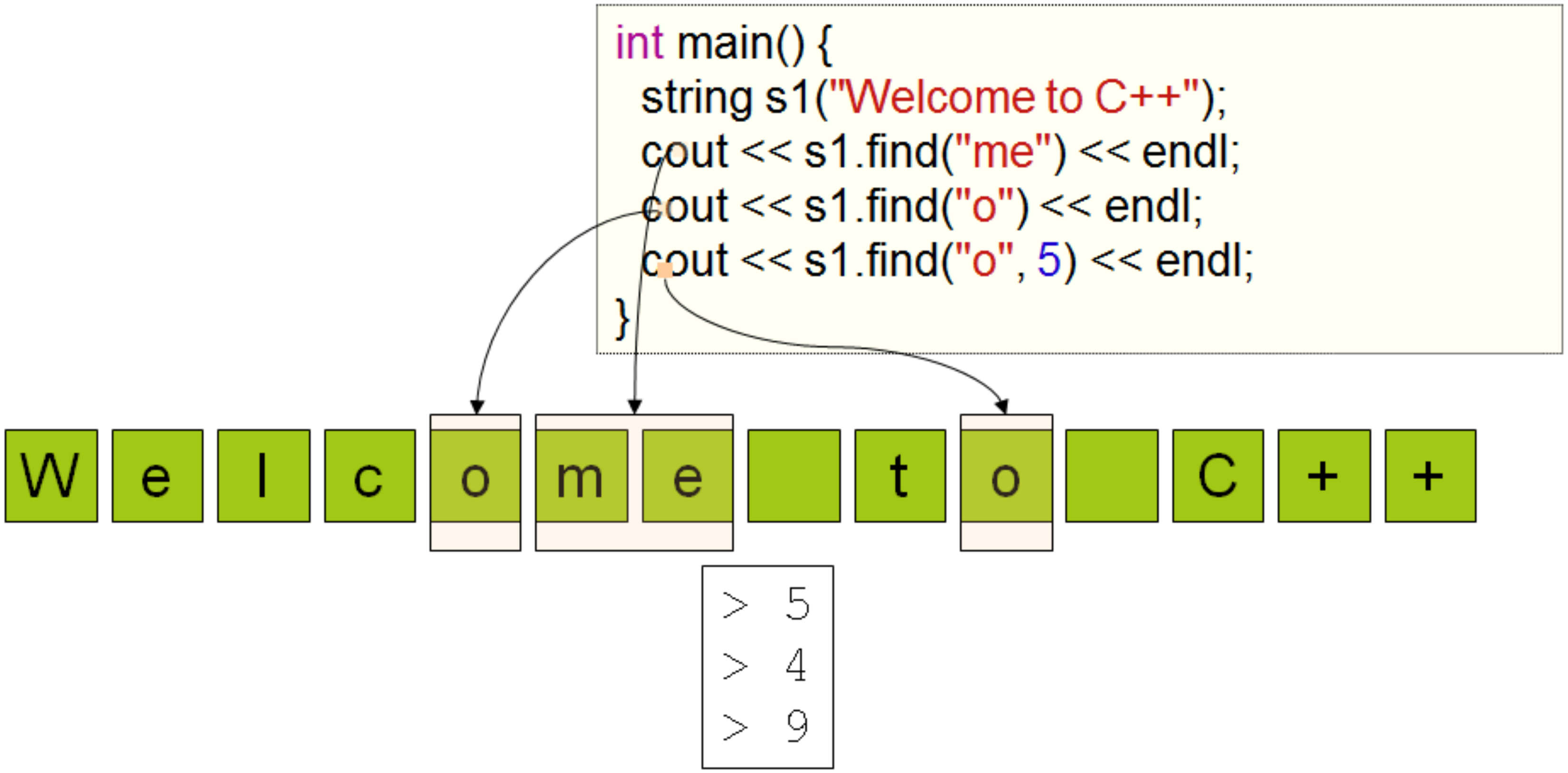
Substrings

```
int main() {
  string s1("Welcome to C++");
  cout << s1.substr(0, 7) << endl;
  cout << s1.substr(5, 5) << endl;
  cout << s1.substr(11, 12) << endl;
}
```



```
> Welcome
> me to
> C++
```

Searching for substrings



```
int main() {
  string s1("Welcome to C++");
  cout << s1.find("me") << endl;
  cout << s1.find("o") << endl;
  cout << s1.find("o", 5) << endl;
}
```

```
> 5
> 4
> 9
```

Insert and Replace

```
int main() {
  string s1("Welcome to C++");
  s1.insert(11, "ANSI ");
  cout << s1 << endl;
  s1.replace(11, 8, "Java");
  cout << s1 << endl;
}
```

```
> Welcome to ANSI C++
> Welcome to Java
```

Accessing individual characters

```
string a = “Hello World”

char c1 = a.at(2);
char c2 = a[2];
```

$\left.\begin{array}{l}\texttt{char c1 = a.at(2);}\\\texttt{char c2 = a[2];}\end{array}\right\}$ Equivalent calls

You may also replace characters using the [ ] notation

Reading strings from user

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

int main() {
  string s1;
  cout << “Enter a string:  “;
  getline(cin, s1);
  cout << s1 << endl;
}
```

Don’t use cin.getline!!!
That is only for C-Strings

C-strings or strings?

In most use cases, you'll be far better off using **strings**

C-strings are actually "behind the scenes" when using the string object type

Occasionally, you will need to use C-strings when interacting with older code, or with C code.

Lab 12

Write a program that produces statistics about a given sentence that the user types.

- Ask the user for a string
- Print out number of vowels, consonants, and words found in the string.
    - Note: use the isalpha function to determine if it a character is a letter.
    - use a switch to determine if the letter is a vowel
    - Words could be separated by more than one space…
        - “   This   sentence has   five words.”

Lab 12 - Recommendation

TEST 3 Functions completely independently!

```
bool isVowel(char c);
```

Write a function accepts a character as a parameter, and returns true if a vowel, false if anything else.

```
int countVowels(string str);
```

Write a function that returns number of vowels in a word. Do this by checking each character, and if the character is a letter (isalpha), then check if its a vowel (isVowel)

```
int countConsonants(string str)
```

Do the same thing as countVowel… but check for letters that are NOT vowels.

```
int countWords(string str);
```

Write a function that scans across each character. If it is a non-space (isspace), and its either the first character, or it comes immediately after a space (isspace), then it represents a word. Return number of words.

Lab 12 - Recommendation

Once your functions have been tested, write the “real” main program:

Read string from user.

Call countVowels(), print result.

Call countConsonants(), print result.

Call countWords, print result...