Announcements

- HW1 is up - due next Wednesday
## Data Types

<table>
<thead>
<tr>
<th>C++ Type</th>
<th>Description</th>
<th>Literals</th>
<th>Python analog</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>logical value</td>
<td>true</td>
<td>bool</td>
</tr>
<tr>
<td>short</td>
<td>integer (often 16 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>integer (often 32 bits)</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>long</td>
<td>integer (often 32 or 64 bits)</td>
<td>39L</td>
<td>int</td>
</tr>
<tr>
<td>--------</td>
<td>integer (arbitrary-precision)</td>
<td></td>
<td>long</td>
</tr>
<tr>
<td>float</td>
<td>floating-point (often 32 bits)</td>
<td>3.14f</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>floating-point (often 64 bits)</td>
<td>3.14</td>
<td>float</td>
</tr>
<tr>
<td>char</td>
<td>single character</td>
<td>‘a’</td>
<td></td>
</tr>
<tr>
<td>string</td>
<td>character sequence</td>
<td>“Hello”</td>
<td>str</td>
</tr>
</tbody>
</table>

*Note: In C++, `string` is actually a class and objects of the `string` class must be instantiated with the `new` operator. The `string` class is a part of a library and you must use the `#include` directive to include it in your code. Use less space in the table.*
Data Types (cont.)

- Each integer type can also be unsigned. Instead of ranging from \(-2^{b-1}\) to \(2^{b-1}-1\),
goes from 0 to \(2^b-1\).
Char versus String

```c
char a;
a = 'a';
a = 'h';
```

```c
String word;
word = "CS 180";
```

Strings are not automatically included! They are standard in most libraries, but need to be imported that library.

---

At top of file:

```c
#include <string>
using namespace std;
(to import)
```
## Strings

Lots of functions (similar to Python)

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.size()</code></td>
<td>Either form returns the number of characters in string <code>s</code>.</td>
</tr>
<tr>
<td><code>s.length()</code></td>
<td></td>
</tr>
<tr>
<td><code>s.empty()</code></td>
<td>Returns <code>true</code> if <code>s</code> is an empty string, <code>false</code> otherwise.</td>
</tr>
<tr>
<td><code>s[index]</code></td>
<td>Returns the character of string <code>s</code> at the given index</td>
</tr>
<tr>
<td></td>
<td>(unpredictable when <code>index</code> is out of range).</td>
</tr>
<tr>
<td><code>s.at(index)</code></td>
<td>Returns the character of string <code>s</code> at the given index</td>
</tr>
<tr>
<td></td>
<td>(throws exception when <code>index</code> is out of range).</td>
</tr>
<tr>
<td><code>s == t</code></td>
<td>Returns <code>true</code> if strings <code>s</code> and <code>t</code> have same contents, <code>false</code> otherwise.</td>
</tr>
<tr>
<td><code>s &lt; t</code></td>
<td>Returns <code>true</code> if <code>s</code> is lexicographical less than <code>t</code>, <code>false</code> otherwise.</td>
</tr>
<tr>
<td><code>s.compare(t)</code></td>
<td>Returns a negative value if string <code>s</code> is lexicographical less than <code>t</code>,</td>
</tr>
<tr>
<td></td>
<td>zero if equal, and a positive value if <code>s</code> is greater than <code>t</code>.</td>
</tr>
<tr>
<td><code>s.find(pattern)</code></td>
<td>Returns the least index (greater than or equal to index <code>pos</code>, if given),</td>
</tr>
<tr>
<td><code>s.find(pattern, pos)</code></td>
<td>at which <code>pattern</code> begins; returns <code>string::npos</code> if not found.</td>
</tr>
<tr>
<td><code>s.find(pattern, pos)</code></td>
<td></td>
</tr>
<tr>
<td><code>s.find_first_of(charset)</code></td>
<td>Returns the least index (greater than or equal to index <code>pos</code>, if given)</td>
</tr>
<tr>
<td><code>s.find_first_of(charset, pos)</code></td>
<td>at which a character of the indicated string <code>charset</code> is found; returns <code>string::npos</code> if not found.</td>
</tr>
<tr>
<td><code>s.find_last_of(charset)</code></td>
<td>Returns the greatest index (less than or equal to index <code>pos</code>, if given)</td>
</tr>
<tr>
<td><code>s.find_last_of(charset, pos)</code></td>
<td>at which a character of the indicated string <code>charset</code> is found; returns <code>string::npos</code> if not found.</td>
</tr>
<tr>
<td><code>s + t</code></td>
<td>Returns a concatenation of strings <code>s</code> and <code>t</code>.</td>
</tr>
<tr>
<td><code>s.substr(start)</code></td>
<td>Returns the substring from index <code>start</code> through the end.</td>
</tr>
<tr>
<td><code>s.substr(start, num)</code></td>
<td>Returns the substring from index <code>start</code>, continuing <code>num</code> characters.</td>
</tr>
<tr>
<td><code>s.c_str()</code></td>
<td>Returns a C-style character array representing the same sequence of characters as <code>s</code>.</td>
</tr>
</tbody>
</table>
Mutable versus immutable

**Def:** mutable - can be changed
- list
  - `my_list[2] = "word"`

**Def:** immutable - fixed value
- String
  - (can't change letter in a string)
C++ - Maximum Flexibility

In C++, everything is mutable!

```cpp
string word;
word = "hello";
word[0] = "J";

word is now "Jello"
```
Creating variables - a few examples in C++

- All variables must be declared and given a type:
  - `int number;`
  - `int a, b;`  
  - `int age (40);`
  - `int age (current Year - birth Year);`
  - `int age (40);`, `zipcode (63116);`
  - `string greeting ("Hello");`

Note: `int a, string b;` - error
Yes: `int a, string b;`
Forcing things to be immutable:

In some situations, there will be data that we want to be fixed.

To do this, use const:

```plaintext
const float gravity(9.8);
```

Later:
```
gravity = 12;  // compiler will give an error
```
Converting between types:

Be careful! C++ cares about type

```cpp
int a(5);
da is 5

double b;
b = a;
da is 5.0

allowed

char a('w');
int b = a;
c is 5

char letter = 'x';
b = a + x;
c is 5
```

```cpp
int a;
double b(2.67);
b = a;
```

```cpp
a is 2.67
```

(a can't go between strings & #s at all although chars are given their ASCII value)
Control Structures

C++ has loops, conditionals, functions, and objects.

Syntax is similar — but usually just different enough to get you into trouble, also...
While loops

while (bool) {
    body;
}

Note: - bool is any boolean exp: a < b
- don't need :3 if only one command in body:

while (a < b)
    a++;

\[ x = -5; \]
\[ \text{while } (x < 0) \end{while} \]
\[ x = x + 1; \]
\[ \text{cout} \ll x \ll \text{endl;} \]
Defining a function: example

Remember our countdown function from 150?

```cpp
void countdown() {
    for (int count = 10; count > 0; count--)
        cout << count << endl;
}
```

Or with optional parameters:

```cpp
void countdown(int start=10, int end=1) {
    for (int count = start; count >= end; count--)
        cout << count << endl;
}
```
In class exercise!
Go to webpage.

cp -R /Public/chambers/180/exercise

cd exercise

[Password] — to change your password