Data Structures
- Dr. Erin Wolf Chambers

Intro
Today

- Syllabus
- Dive in to C++

Resources you'll use:
- Web page
- Recommended text
- cplusplus.com
- transition guide
This course:
Data structures in C++

First: data structures

What is a data structure?
Container to hold data

¢ along w/ specified ways to interact w/data

Examples:
¢ Array
¢ Lists
¢ Dictionaries
¢ Matrices
¢ Trees
¢ Graphs
¢ Search Trees
¢ Heaps
¢ Etc.
Why should you care?
- You'll use them constantly!
These are many ways to solve a problem.

Goals:

1. Correct
2. Fast
3. Efficient

Choice of data structure is key!

Also: job interviews!
Second: C++ (versus Python)

High level

Python

English-like

Command line running

Interpreted vs. Compiled

Interpreted

More details in code

Interpreted

Low level

Compiled

Interpreted

Interpreted

$2$ phase process

Dynamic vs static typing

Dynamic

\[ \begin{align*}
    \text{int } x &= 5 \\
    \text{int } x &= 5 \\
    \text{string } x &= \text{"cat"}
\end{align*} \]

Static typing

\[ \begin{align*}
    \text{int } x &= \text{"cat"} \\
    \text{Error}
\end{align*} \]
Why should you learn C++?

- faster
- ubiquitous
- need to understand low level details (sometimes)
- more control
Figure 1: Programs for computing a greatest common divisor, as written in Python and C++. 

```python
def gcd(u, v):
    # we will use Euclid's algorithm
    # for computing the GCD
    while v != 0:
        r = u % v  # compute remainder
        u = v
        v = r
    return u

if __name__ == '__main__':
a = int(input('First value: '))
b = int(input('Second value: '))
print('gcd:', gcd(a, b))
```

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

int gcd(int u, int v) {
    /* We will use Euclid's algorithm
     * for computing the GCD */
    int r;
    while (v != 0) {
        r = u % v;  // compute remainder
        u = v;
        v = r;
    }
    return u;
}

int main() {
    int a, b;
    cout << "First value: ";
    cin >> a;
    cout << "Second value: ";
    cin >> b;
    cout << "gcd: " << gcd(a, b) << endl;
    return 0;
}
```
First: While space returns, tabs, etc — all ignored in C++ (big difference from Python)

```cpp
int gcd(int u, int v) { int r; while (v != 0) { r = u % v; u = v; v = r; } return u; }
```

So control structures marked with ( ) and { }, at lines end with ;
<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>false</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>integer (often 32 bits)</td>
<td>39</td>
<td>int</td>
</tr>
<tr>
<td>long</td>
<td>integer (often 32 or 64 bits)</td>
<td>39L</td>
<td>long</td>
</tr>
<tr>
<td>——</td>
<td>integer (arbitrary-precision)</td>
<td></td>
<td></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(^a)</td>
<td>character sequence</td>
<td>&quot;Hello&quot;</td>
<td>str</td>
</tr>
</tbody>
</table>

Figure 2: The most common primitive data types in C++.

\(^a\)Not technically a built-in type; included from within standard libraries.
<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 s.</td>
</tr>
<tr>
<td><code>s.length()</code></td>
<td>Either form returns the number of characters in string s.</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 s at the given index (unpredictable when index is out of range).</td>
</tr>
<tr>
<td><code>s.at(index)</code></td>
<td>Returns the character of string s at the given index (throws exception when index is out of range).</td>
</tr>
<tr>
<td><code>s == t</code></td>
<td>Returns <code>true</code> if strings s and t have same contents, <code>false</code> otherwise.</td>
</tr>
<tr>
<td><code>s &lt; t</code></td>
<td>Returns <code>true</code> if s is lexicographical less than t, <code>false</code> otherwise.</td>
</tr>
<tr>
<td><code>s.compare(t)</code></td>
<td>Returns a negative value if string s is lexicographical less than string t, zero if equal, and a positive value if s is greater than t.</td>
</tr>
<tr>
<td><code>s.find(pattern)</code></td>
<td>Returns the least index (greater than or equal to index pos, if given), at which pattern begins; returns <code>string::npos</code> if not found.</td>
</tr>
<tr>
<td><code>s.find(pattern, pos)</code></td>
<td>Returns the least index (greater than or equal to index pos, if given) at which a character of the indicated string charset is found; returns <code>string::npos</code> if not found.</td>
</tr>
<tr>
<td><code>s.find_first_of(charset)</code></td>
<td>Returns the least index (greater than or equal to index pos, if given) at which a character of the indicated string charset is found; returns <code>string::npos</code> if not found.</td>
</tr>
<tr>
<td><code>s.substr(start)</code></td>
<td>Returns the substring from index start through the end.</td>
</tr>
<tr>
<td><code>s.substr(start, num)</code></td>
<td>Returns the substring from index start, continuing num 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 s.</td>
</tr>
</tbody>
</table>

**Figure 3:** Nonmutating behaviors supported by the `string` class in C++.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s[index] = newChar</code></td>
<td>Mutates string s by changing the character at the given index to the new character (unpredictable when index is out of range).</td>
</tr>
<tr>
<td><code>s.append(t)</code></td>
<td>Mutates string s by appending the characters of string t.</td>
</tr>
<tr>
<td><code>s += t</code></td>
<td>Same as <code>s.append(t)</code>.</td>
</tr>
<tr>
<td><code>s.insert(index, t)</code></td>
<td>Inserts copy of string t into string s starting at the given index.</td>
</tr>
<tr>
<td><code>s.insert(index, num, c)</code></td>
<td>Inserts num copies of character c into string s starting at the given index.</td>
</tr>
<tr>
<td><code>s.erase(start)</code></td>
<td>Removes all characters from index start to the end.</td>
</tr>
<tr>
<td><code>s.erase(start, num)</code></td>
<td>Removes num characters, starting at given index.</td>
</tr>
<tr>
<td><code>s.replace(index, num, t)</code></td>
<td>Replace num characters of current string, starting at given index, with the first num characters of t.</td>
</tr>
</tbody>
</table>

**Figure 4:** Mutating behaviors supported by the `string` class in C++.