CS 180: Intro to C++

Announcements

- Syllabus
- Lab tomorrow
- HW 1 soon
Resources for this class

- Text book
- Transition guide (look for pdf on webpage)
- cplusplus.com
- Tutoring + office hours
This course: data structures in C++

First, C++. (More on that next.)

But — what is a data structure?

Container for data

plus constrained way to interact

- trees (sorted)
- list
- dictionary
- array
- set
- tuple
- heaps
- graphs
Why you should care about them:
- Many ways to solve a problem

Goals:
1. Correct
2. Fast
3. Efficient $\rightarrow$ space

$\rightarrow$ Data structure choice is key!
(And you will use them!)
C++ versus Python
High level versus low level.
- readable
- closer to machine code

Interpreted versus compiled.
- Compile
- then run executable

Dynamic versus static typing
- int x;
- x = 5;
- x = "Hello";
Why learn C++?
- faster
- ubiquitous
- understand low level details
- control
Comparison

Python

```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))
```

C++

```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;
}
```
White space
returns, tabs, etc. are ignored in C++

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

(Recall that these were very important in python)

Here, we use () and {} to mark loops, booleans, etc.
Compiling

In Python, you save code as `gcd.py` & then type "python gcd.py" to run it.

In C++:
* Save as `gcd.cpp`
* Type "g++ -o gcd gcd.cpp"
* Type "./gcd"
## Data Types

<table>
<thead>
<tr>
<th>C++ Type</th>
<th>Description</th>
<th>Literals</th>
<th>Python analog</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool</code></td>
<td>logical value</td>
<td><code>true</code> <code>false</code></td>
<td><code>bool</code></td>
</tr>
<tr>
<td><code>short</code></td>
<td>integer (often 16 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>int</code></td>
<td>integer (often 32 bits)</td>
<td><code>39</code></td>
<td></td>
</tr>
<tr>
<td><code>long</code></td>
<td>integer (often 32 or 64 bits)</td>
<td><code>39L</code></td>
<td><code>int</code></td>
</tr>
<tr>
<td></td>
<td>integer (arbitrary-precision)</td>
<td></td>
<td><code>long</code></td>
</tr>
<tr>
<td><code>float</code></td>
<td>floating-point (often 32 bits)</td>
<td><code>3.14f</code></td>
<td></td>
</tr>
<tr>
<td><code>double</code></td>
<td>floating-point (often 64 bits)</td>
<td><code>3.14</code></td>
<td><code>float</code></td>
</tr>
<tr>
<td><code>char</code></td>
<td>single character</td>
<td><code>'a'</code></td>
<td></td>
</tr>
<tr>
<td><code>string</code></td>
<td>character sequence</td>
<td>&quot;Hello&quot;</td>
<td><code>str</code></td>
</tr>
</tbody>
</table>
Data Types (cont)

- Ints can also be unsigned: instead of ranging from \(-(2^{b-1})\) to \((2^{b-1}-1)\), go from 0 to \(2^{(b-1)}\).

- Strings and chars are very different.
Char versus string

```c
#include <string>
char a;
a = 'a';
a = 'h';
```

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

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