Announcements

- Extra credit for help session - today at 4pm

- Lab tomorrow (will be short)

- HW out tonight or tomorrow, due 1 week
Comparison

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++

#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++.

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

This is not acceptable to submit (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` and then type "python gcd.py" to run it.

Later on: makefiles

- In C++:
  - Save as `gcd.cpp`
  - Type "g++ -o gcd gcd.cpp"
  - Type "./gcd"  
    - Optional: if omitted, defaults to "a.out"
## 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></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><code>int</code></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><code>float</code></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><code>str</code></td>
</tr>
<tr>
<td><code>string</code></td>
<td>character sequence</td>
<td><code>&quot;Hello&quot;</code></td>
<td><code>str</code></td>
</tr>
</tbody>
</table>

- **numerals**: `a < b` smaller
- **single quotes**: `'"' single quotes` 'double'
- **are not a default data type**
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.

```
char a;
int a = 'a';
```

```cpp
#include <string>

using namespace std;
```

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

Strings are not automatically included. Standard in most libraries, but need to import.
Strings

<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 strings <code>s</code> at the given index</td>
</tr>
<tr>
<td></td>
<td>(unpredictable when index is out of range).</td>
</tr>
<tr>
<td><code>s.at(index)</code></td>
<td>Returns the character of strings <code>s</code> at the given index</td>
</tr>
<tr>
<td></td>
<td>(throws exception when index 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 string <code>t</code>, 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), at which pattern begins; returns <code>stringnpos</code> if not found.</td>
</tr>
<tr>
<td><code>s.find(pattern, pos)</code></td>
<td></td>
</tr>
<tr>
<td><code>s.rfind(pattern)</code></td>
<td>Returns the greatest index (less than or equal to index <code>pos</code>, if given), at which pattern begins; returns <code>stringnpos</code> if not found.</td>
</tr>
<tr>
<td><code>s.rfind(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), at which a character of the indicated string <code>charset</code> is found; returns <code>stringnpos</code> if not found.</td>
</tr>
<tr>
<td><code>s.find_first_of(charset, pos)</code></td>
<td></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), at which a character of the indicated string <code>charset</code> is found; returns <code>stringnpos</code> if not found.</td>
</tr>
<tr>
<td><code>s.find_last_of(charset, pos)</code></td>
<td></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

lists

**Def**: immutable: can't be changed

string

word [X]
C++: Maximum flexibility

Everything is mutable by default!

```cpp
string word;
word = "Hello";
word[0] = 'J';

"Jello"
```
Creating variables: create all at beginning of function.

All variables must be explicitly created and given a type.

```java
int number;
int a, b;
int age (35);
int age2 (currYear - birthYear);
int age3 (21), zipcode (63116);
String greeting ("Hello");
```
Immutable variables

We can force some variables to be immutable—use const:

```javascript
class float gravity (-9.8)
```

Why?
- ease of testing
- forces the value to stay fixed
Converting between types

Be careful!

```cpp
int a(5);
double b;
b = a;  \text{\textcolor{red}{b = 5.0}}
```

```cpp
int a;
double b (2.67);
a = b \leftarrow \text{trunc}
```

```cpp
a = b + .5 \leftarrow \text{round}
```
Converting with strings

- Can't go between strings and numeric types at all. "27" is not a number.
- But chars will convert to numbers. How?

ASCII codes

int number = int(letter);
Control Structures

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

Syntax is similar, but just slightly different enough to get into trouble.

(Remember to use your book’s index in a pinch!)
While loops

\[
\text{while (bool) } \\
\quad \text{ body; } \\
\quad \text{// end of while (bool)}
\]

Notes:
- bool is any boolean expression
- don't need \texttt{;} if only 1 command in the loop: \texttt{while (a<b) \quad while (a<b) \quad a++; b = a; \quad a++;}
**Defining a function:** example

Remember countdown function from 150?

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

```cpp
void countdown (int start, int finish) {
    ...
}
```
Optional arguments

```c
void countdown(int start=10, int end=1) {
    for (int count = start; count >= end; count--)
        cout << count << endl;
}
```
Tomorrow - lab

Friday - finish control structures