Announcements

- HW1 due next Wednesday
- Program 1 will be out today
due in 2 weeks

- will be in pairs
- 10% of credit is for "early checkpoint"
Functions: Basic Structure

```
returntype functionName (type parameter1, ...)
```

```
body;
```

Example:

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

Every program defaults to running a special "main" function first.

(In python, we just started typing code.)

```python
int main()
    
    body;
```
Input + Output

C++ has several predefined useful classes.

<table>
<thead>
<tr>
<th>Class</th>
<th>Purpose</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>istream</td>
<td>Parent class for all input streams</td>
<td>&lt;iostream&gt;</td>
</tr>
<tr>
<td>ostream</td>
<td>Parent class for all output streams</td>
<td>&lt;iostream&gt;</td>
</tr>
<tr>
<td>iostream</td>
<td>Parent class for streams that can process input and output</td>
<td>&lt;iostream&gt;</td>
</tr>
<tr>
<td>ifstream</td>
<td>Input file stream</td>
<td>&lt;fstream&gt;</td>
</tr>
<tr>
<td>ofstream</td>
<td>Output file stream</td>
<td>&lt;fstream&gt;</td>
</tr>
<tr>
<td>fstream</td>
<td>Input/output file stream</td>
<td>&lt;fstream&gt;</td>
</tr>
<tr>
<td>istringstream</td>
<td>String stream for input</td>
<td>&lt;sstream&gt;</td>
</tr>
<tr>
<td>ostringstream</td>
<td>String stream for output</td>
<td>&lt;sstream&gt;</td>
</tr>
<tr>
<td>stringstream</td>
<td>String stream for input and output</td>
<td>&lt;sstream&gt;</td>
</tr>
</tbody>
</table>

(We'll use iostream & fstream the most.)
Using cout + cin (in iostream)

#include <iostream>
using namespace std;

otherwise: std::cin

Notes:
- gets cout + cin
- separate distinct variables by >> or <<
  cout    cin
- use endl instead of "\n"
Examples

```python
1 print "Hello"
2 print
3 print "Hello,", first
4 print first, last  # automatic space
5 print total
6 print str(total) + "."  # no space
7 print "Wait...",  # space; no newline
8 print "Done"
```

```cpp
1 cout << "Hello" << endl;
2 cout << endl;
3 cout << "Hello, " << first << endl;
4 cout << first << " " << last << endl;
5 cout << total << endl;
6 cout << total << "." << endl;
7 cout << "Wait... ";  // no newline
8 cout << "Done" << endl;
```

Figure 7: Demonstration of console output in Python and C++. We assume that variables first and last have previously been defined as strings, and that total is an integer.
Formatting output

Unfortunately, '%d' output is not really available.
(Inherited from C, so there but can't be used with C++ objects like strings.)

Python

```python
print '%s: ranked %d of %d teams' % (team, rank, total)
```

C++

```cpp
cout << team << ": ranked " << rank << " of " << total << " teams" << endl;
```
Setting precision is harder:

```c++
print "pi is \%0.3f\"\% pi
output?
```

```
\textbf{pi is 3.141}
```

In C++:

```c++
cout << "pi is \"\" fixed << setprecision(3)
<< pi << endl;
```

\textbf{Note:} Precision stays set to 3.
Input: Strings

Python: raw_input

```
person = raw_input('What is your name?')
```

C++: cin + getline

```
string person;
cout << "What is your name? ";
getline(cin, person);
```

Note (for getline):
- Inputs a *String*
- Stores up to the newline, but strips the newline off
Cin : Other data types

Python:

```python
number = int(raw_input('Enter a number from 1 to 10: '))
```

C++:

```cpp
int number;
cout << "Enter a number from 1 to 10: ";
cin >> number;
```

Note:
- don’t need to cast
- needs to be of correct type!
Some other differences with cin:

Chaining multiple inputs

```cpp
int a, b;
cout << "Enter two integers: ";
cin >> a >> b;
cout << "Their sum is " << a + b << "." << endl;
```

Note: - different types are allowed
  (but must match the variable)

- separated by any whitespace!

Ex: : 10 20 "\n"
```cpp
10
20
```
```
A word of caution:

Ex:

```cpp
string person;
cout << "What is your name? ";
cin >> person;
```

I type "Erin Wolf Chambers/n".

What happens?

```cpp
person = "Erin"
```

If you want everything up to newline use getline(cin, person)
Another caution:

```cpp
int age;
string food;
cout << "How old are you? ";
cin >> age;
cout << "What would you like to eat? ";
getline(cin, food);

> 30
pizza
stream: 30

age = 30
food = ""
```
File Streams: Input

If file name is known:

```cpp
ifstream mydata("scores.txt");
```

If file name is unknown:

```cpp
ifstream mydata;
string filename;
cout << "What file? ";
cin >> filename;
mydata.open(filename.c_str());
```

- Input to open a file needs to be a C-style string
- Use `c_str()` to cast to C-style string
Output:
By default, opening ofstream overwrites an existing file!
(just like "w" option in Python)

To append:
ofstream datastream("scores.txt", ios::app);

just like 'a' in Python
There is also an "fstream" object which allows both input & output.

Much more confusing.

We've used ifstream & ofstream.
String Streams

Casting from numbers to strings is not straightforward.

```cpp
int age(40);
string displayedAge;
stringstream ss;
ss << age; // insert the integer representation into the stream
ss >> displayedAge; // extract the resulting string from the stream
```

Can’t just type:

```cpp
displayedAge = string(age); // ERROR in C++
```
#include <string>

Classes

Creating an instance of a class

```cpp
string s;
string greeting("Hello");
```

**NEVER**: `string s();`

*Why?* Creates empty function called `s` that returns a string

**NEVER**: `string("Hello") greeting;`

*Why?* Gives an error
Defining a class: Remember the Point class?

class Point {
private:
  double _x;
  double _y;

public:
  Point() : _x(0), _y(0) {}
  // constructor
  double getX() const {
    return _x;
  }
  // accessor
  void setX(double val) {
    _x = val;
  }
  // mutator
  double getY() const {
    return _y;
  }
  // accessor
  void setY(double val) {
    _y = val;
  }
  // mutator
};
  // end of Point class (semicolon is required)
Classes - differences:

- Data (public or private) is explicitly declared, not just used in constructor.

- Constructor:
  - has same name as class
  - initializes data from the class

Point (c) \( \mathcal{E} \)

- \( x = 0 \)
- \( y = 0 \)
- \( z \)
A more complicated constructor:

Point(double initialX=0.0, double initialY=0.0) : _x(initialX), _y(initialY) { }

- Allows default parameters,
  but body is still empty,
Other things to note:

3. No `self`! Can just use `-x` or `-y`, or understood to be attributes of current object.

(Could use this, i.e. this `-x`, if necessary.)

4. Access control - public versus private

```
mypoint. -x  # give an error if in main
```
Other things to note (cont):

5 accessor versus mutator:

```cpp
double getX( ) const {
  return x;
}  // accessor

-x = 5; gives error
```

```cpp
void setX(double val) {
  x = val;
}  // mutator
```

Forced by compiler: