CS 180 - More C++ info

Note Title 8/26/2010

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

or void

returntype functionName (type parameter1, ...)

{

body;

}

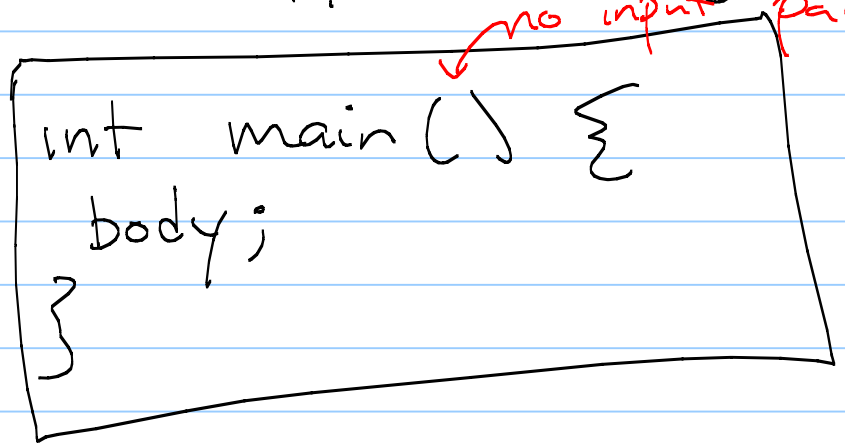
Example:

```
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.)



Input + Output

C++ has several predefined, useful classes.

Class	Purpose	Library
istream	Parent class for all input streams	<iostream>
ostream	Parent class for all output streams	<iostream>
iostream	Parent class for streams that can process input and output	<iostream>
ifstream	Input file stream	<fstream>
ofstream	Output file stream	<fstream>
fstream	Input/output file stream	<fstream>
istringstream	String stream for input	<sstream>
ostringstream	String stream for output	<sstream>
stringstream	String stream for input and output	<sstream>

occasional

(We'll use iostream & fstream the most.)

Using cout & cin (in iostream)

at top of file

```
#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"
```

C++

```
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

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

C++

```
cout << team << ": ranked " << rank << " of " << total << " teams" << endl;
```

Setting precision is harder:

print 'pi is %.3f' % pi

output?

pi is 3.141

In C++:

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

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:

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

C++:

```
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 diffences with cin:

Chaining multiple inputs

```
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"

10\n

20\n

A word of caution:

Ex:

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

I type "Erin Wolf Chambers /n".

What happens?

person = "Erin"

if you want everything up to newline use getline(cin, person)

Another caution:

```
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\npizza\n

age = 30
food = ""

File Streams: Input

If file name is known:

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

If file name is unknown:

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

filename is a C++ string

- 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

fstream

There is also an "fstream" object which allows both input & output.

Much more confusing.

We've used ifstream & ofstream

ifstream: input

ofstream: output

String Streams

Casting from numbers to strings is not straightforward.

```
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:
displayedAge = String(age);
ERROR in C++

Classes

#include <string>

Creating an instance of a class

```
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;                    // explicit declaration of data members
    double _y;

  public:
    Point( ) : _x(0), _y(0) { }   // constructor

    double getX( ) const {        // accessor
      return _x;
    }

    void setX(double val) {       // mutator
      _x = val;
    }

    double getY( ) const {        // accessor
      return _y;
    }

    void setY(double val) {       // mutator
      _y = val;
    }

};                                // end of Point class (semicolon is required)
```

private data is enforced! can't be used outside the class

Classes - differences:

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

2. Constructor!
   - has same name as class
   - initializes data from the class

```
Point () {
    -x = 0;
    -y = 0;
}
```

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, & understood to be attributes of current object.

(Could use this, ie 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:

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

_x = 5; gives error

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

Forced by compiler: