CS180: Lecture 3

Note Title 8/27/2009

Announcements

- HW 1 - due next Friday by start of class (Email preferred.)
- Will need textbook in 1-2 weeks

Note Title 8/25/2009

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>

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

Using cout & cin

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

→ loads standard input/output

Notes: - gets cout & cin
- Separate distinct variables by >> or <<
  - >> → cin
  - << → cout
- 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

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

person must be a string

Note (for getline):

- inputs a string
- stores up to the newline (but strips off newline)

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

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:
- a & b can have different types
- separates by white space

42 ␣ 63 enter

42 enter 63 enter

A word of caution:

Ex:

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

If I type:

Erin Wolf Chambers -enter

value of person is "Erin"

still waiting

Another caution:

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

40 enter pizza enter

age = 40 ← "\n pizza \n"

food = ""

↑ empty string

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( ));
```

← must be a 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);
```

fstream

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

Much more confusing.

Classes

Creating an instance of a class

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

Never: `string s( );`

Why? creates an empty fcn whose return type is string

Never: `string("Hello") greeting;`

Why? give 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)
```

/* Constructor */

Classes - differences:

1. Data (public or private) is explicitly declared, not just used in constructor.
2. Constructor:
   - no return value
   - name of class
   - initializer list -x(0) -y(0) (must initialize variables from declaration)
   - empty body {}

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

in main:

```
Point mypoint;
mypoint._x = 3;      <- error
mypoint.setX(3);
```

Other things to note (cont):

(5) accessor versus mutator:

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

difference?

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

Forced by compiler:

if const appears in fcn declaration, any attempt to change member data will give a compile error

Robust Point Class:

```
class Point {
  private:
    double _x;
    double _y;

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

    double getX( ) const { return _x; }      // same as simple Point class
    void setX(double val) { _x = val; }      // same as simple Point class
    double getY( ) const { return _y; }      // same as simple Point class
    void setY(double val) { _y = val; }      // same as simple Point class

    void scale(double factor) {
      _x *= factor;
      _y *= factor;
    }

    :
```

Robust Point class cont:

```
    double distance(Point other) const {
      double dx = _x - other._x;
      double dy = _y - other._y;
      return sqrt(dx * dx + dy * dy);          // sqrt imported from cmath library
    }

    void normalize( ) {
      double mag = distance( Point( ) );       // measure distance to the origin
      if (mag > 0)
        scale(1/mag);
    }

    Point operator+(Point other) const {
      return Point(_x + other._x, _y + other._y);
    }

    Point operator*(double factor) const {
      return Point(_x * factor, _y * factor);
    }

    double operator*(Point other) const {
      return _x * other._x + _y * other._y;
    }
};   // end of Point class (semicolon is required)
```

differences?

Things to note:

- `-x + other.-x` ← allowed if inside the class

- using operator+, will be `x + y`
- two versions of *

Additional functions
(Not in class)

```
// Free-standing operator definitions, outside the formal Point class definition
Point operator*(double factor, Point p) {
  return p * factor;                          // invoke existing form with Point as left operand
}

ostream& operator<<(ostream& out, Point p) {
  out << "<" << p.getX( ) << "," << p.getY( ) << ">";       // display using form <x,y>
  return out;
}
```