# Data Structures

## I/O Classes

# Announcements & Reminders

- Everyone should be able to use linux lab!
- HW - due tomorrow
- Lab tomorrow
  (prelab due before 9am tomorrow)
- Wed. office hours will be moving
  (stay tuned)

# Last time

- Control structures (loops, if, fcns,..)
- Arrays (+ seg faults)
- Began I/O

| 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> |

Figure 6: Various input and output stream classes.

# Formatting I/O : [#include <iostream> Using namespace std;]

**Python**

```
1  print "Hello"
2  print                          # blank line
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;                  // blank line
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.

## Setting precision:

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

3.141

↳ This stays!

```
cout << setw(10) << item << " " << setw(5) << quantity << endl;
```

This is equivalent to the Python command print '%10s %5d' % (item, quantity). If we execute this command once with values pencil and 50, and then with values pen and 100, the output is aligned as:

```
    pencil    50
       pen   100
```

# Using cin

```
int number;
cout << "Enter an integer: ";
cin >> number;
```

# Notes

- Inputs are separated by any whitespace!

```
cin >> a >> b;
```

5 _ 10 ↵

(Careful w/ strings!) 5 ↵ 10 ↵

- Type of input must match type of variable (not all strings)

↑ Python

## Issue

```
string person;
cout << "Enter your name: ";
cin >> person;
```

> Erin Chambers

## Fix: use getline:

```
getline(cin, person);
```

cin — input

# Another issue:

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

A typical user session might proceed as follows.

```
How old are you?  42
What would you like to eat? pepperoni pizza
```

istream: 42\n pepperon pizza\n

# File streams : fstream

```
#include <fstream>
using namespace std;
ifstream mydata("scores.txt");
```

ifstream = input stream

mydata = variable name

"scores.txt" = filename in current directory

else: "../scores.txt"

"/public/chamber/lab1/..."

```
int val;
mydata >> val;
```

Adding input:

```
ifstream mydata;
string filename;
cout << "What file? ";
cin >> filename;
mydata.open(filename.c_str( ));      // parameter to open must be a C−style string
```

(legacy from C)

# Outstreams:

ofstream mystream("scores.txt");

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

Note:

creates a file (overwriting if necessary)

→ this appends (Python: "a")

There is an fstream object.

Complex!

(We'll avoid in this class)

# String streams

Ex: cast between # & string

```
int age(42);
string displayedAge;
stringstream ss;
ss << age;              // insert the integer representation into the stream
ss >> displayedAge;     // extract the resulting string from the stream
```

# A note on variable scopes

lifetime of a piece of data

```
int main() {
    int a;
    if (a > 0) {
        int b = 12;
    } // b is destroyed
    else {
        int b = 16;
    } // b is destroyed
    cout << "a is " << a << endl;
    cout << "b is " << b << endl;
} // a is destroyed
```

all initialization done at beginning

Compile error

# Arrays as fcn inputs

Ex: write a fcn to specify if sum of values is even

empty

```
bool sumEven(int anArray[], int size){
    int sum = 0;
    for(int i=0; i<size; i++){
        sum += anArray[i];
    }
    return (sum % 2 == 0);
}
```

```
if (sum % 2 == 0)
    return true;
else
    return false;
```

Here: int a[] actually makes a (the array) a pointer!

More later...

Doesn't copy entire array, just has something "pointing" to start of it.

To call:

```
int main() {
    // create & make array Ar
    //  of size 65
    if(sumEven(Ar, 65))
        cout << "it's even";
    //more code
}
```

# Classes

What is a class?

"object" —
stores data + fcns
that restrict how you
interact w/ data

## Creating one:

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

Never: `string s();` ← initialization or declaration

Why? declared s a function, 0 inputs, returning a string

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

↑ wierd

Making our own:

must capitalize

```
1  class Point {
2  private:
3    double _x;                  // explicit declaration of data members
4    double _y;
5
6  public:
7    Point( ) : _x(0), _y(0) { } // constructor
8
9    double getX( ) const {      // accessor
10     return _x;
11   }
12
13   void setX(double val) {     // mutator
14     _x = val;
15   }
16
17   double getY( ) const {      // accessor
18     return _y;
19   }
20
21   void setY(double val) {     // mutator
22     _y = val;
23   }
24
25 };                             // end of Point class (semicolon is required)
```

means for class only

means visible

constructor

can't change anything

semicolon

Figure 9: Implementation of a simple Point class.