

CSE 143

More Stream I/O

Appendix A

3/25/2001 G-1

Streams as C++ Classes

- Streams are C++ classes
- Streams have lots of built-in methods
- We use the "." syntax to access member functions, as usual.

```
inFile.get(ch);      // get a character
outFile.put(ch);     // put a character
outFile.getline(str, len); //get a whole line
outFile.close();     // close the stream
inFile.eof();        // end of File??
```

3/25/2001 G-2

Stream Classes

- cin and cout are defined in <iostream>.
- Library <fstream> contains similar classes for file I/O
- Input stream classes:
 - istream: console input (cin)
 - ifstream: file input
- Output stream classes
 - ostream: console output (cout, cerr)
 - ofstream: file output

3/25/2001 G-3

Stream Class Relationships

- Every ifstream (file) is also an istream.
 - An ifstream is an "enhanced" istream that has extra capabilities to work with disk files
 - An ifstream object can be used wherever an istream object is needed (function parameter, for example)
- But the reverse is not true. An istream is not also an ifstream.
 - So if an ifstream is explicitly called for, cin can't be used
- A similar relationship holds between ofstream and ostream.
- This is an example of "inheritance"
 - An important object-oriented concept we will study later

3/25/2001 G-4

Reading a Whole Line

- Reading
Seattle Rain
- vs
Seattle-Rain
- cin >> stringvar won't do the former -- why?
- Need an additional function: getline
cin.getline(stringvar, len);
- Dot notation! What's happening here??
 - Answer: Remember, cin and cout are really objects

3/25/2001 G-5

Unformatted Stream I/O

- >> and << provide formatted I/O.
- There are member functions which provide unformatted (character-level) I/O. Examples:

```
char ch; char s[100];
cin.get(ch); // read 1 character into ch
cin.getline(s,n); // read next line into s
cout.put(ch); // write 1 character ch
```
- Variations available to limit how many characters are read, specify end-of-line characters, etc.

3/25/2001 G-6

Stream States (Review)

- All streams have a “state”.
- All streams are objects (instances of stream classes)
- Several member functions are available to check or set state.

```
cin.eof(); // true if cin eof reached
cin.clear(); // set state to “good”
```
- The stream itself can be used in an expression to check its state

```
if (!cin)
    cerr << “error or eof on cin” << endl;
```

3/25/2001 G-7

End-Of-File State

- Means there is no more input in the stream
- *eof is a state; it's not a special value in the stream*
- eof is most often used with files
- eof with keyboard input?
 - User signals by typing a special key combination
 - CNTL-Z, CNTL-D, etc. depends on operating system
 - The special key is NOT sent to the program. The eof status is what is detected.

3/25/2001 G-8

Input Errors

- Stream input “fails” if the next thing in the input has the wrong format or if there is no more data (end of file).
- If an input operation fails, the variable involved is not changed.

```
if (cin >> k)
    cout << “new value for k read ok”;
else
    cout << “input failed; ”
        << “k not changed”;
```

3/25/2001 G-9

Input Errors (cont)

- Once a stream input operation has failed, any further operations will also fail until the stream state is cleared.

```
// suppose next input is “xyz”
cin >> k; // fails (why?); k unchanged
cin >> j; // cin state not good, so
          // nothing happens
cin.clear(); // cin can be used for
             // input again
```

3/25/2001 G-10

Recall: BankAccount class

- We've seen several variations on a bank account class

```
// Representation of a bank account
class BankAccount {
public:
    // create account with given owner
    BankAccount(string name);
    // add amount to account balance
    void deposit(double amount);
    // = current account balance
    double amount();
private:
    string owner; // account holder's name
    double balance; // current account balance
};
```

3/25/2001 G-11

User-Defined Stream I/O

- We would like to define stream I/O for bank accounts so we could do things like this...

```
#include <iostream>
using namespace std;
int main() {
    BankAccount ba(“Lazowska”);
    ...
    ba.deposit(450);
    cout << ba << endl;
}
```

3/25/2001 G-12

Overloading <<

- What's needed is to define the meaning of << for BankAccounts. In essence, we want

```
??? operator<<(ostream &s,
               const BankAccount &b) {
    s << "Account owner is " << b.owner
      << ", balance is " << b.balance;
}
```

- Issues

- How does operator<< access fields of b?
- What is the result type of operator<<?
- operator<< can't be a member function (why?)

3/25/2001 G-13

friend functions

- Sometimes operator<< can be written using only public operations of the class
- If it needs access to private details, declare it in the class as a friend function

```
// Representation of a bank account
class BankAccount {
public:
    ...
private:
    string owner; // account holder's name
    double balance; // current account balance

    friend ??? operator<<(ostream& s,
                        const BankAccount &b);
};
```

- (Still to do: fix result type of function)

3/25/2001 G-14

operator<< result

- The issue is that stream operators are supposed to chain
- Example

```
BankAccount a, b;
...
cout << a << b << endl;
```

- operator<< is left associative, so this means

```
((cout << a) << b) << endl;
```

3/25/2001 G-15

operator<< result

- If we write this out explicitly, it's fairly easy to see that the result needs to refer to the stream somehow

```
operator<< (
    operator<< (
        operator<< (cout, a),
        b),
    endl)
```

- The correct result type is a reference to the type of the stream.

3/25/2001 G-16

Definition of BankAccount <<

- Declare << as a friend function if needed

```
class BankAccount {
private:
    ...
    friend ostream & operator<<(ostream& s,
                               const BankAccount &b);
};
```

- Have operator<< return a reference to the stream

```
ostream & operator<<(ostream &s,
                   const BankAccount &b) {
    s << "Account owner is " << b.owner
      << ", balance is " << b.balance;
    return s;
}
```

return reference to the stream

3/25/2001 G-17

Notes and Advice

- File and stream processing can get VERY baroque
 - Many details, gotchas, exceptions, etc. in the C++
 - File formats are often complex
- Learn the basics
- Try to keep it simple (not always possible)
- You can't memorize it all
- Buy a good C++ book and keep it handy when programming!
 - Bookstore has lots to choose from. Browse and buy one you like

3/25/2001 G-18