Stream I/O
[Appendix C]

Input/Output Concepts

- New syntax in C++, but same fundamental concepts:
  - input vs. output, read vs. write
  - files, file name vs. file variable
  - open, close
  - end-of-file
  - interpreting sequences of characters as C/C++ types

Stream I/O

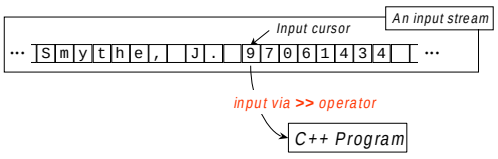
- C++ uses the stream abstraction for I/O
  - both for keyboard/monitor and for files
- C++ can also use old C-style printf, scanf, etc.
  - Mixing the two is not recommended
  - You must use **only** stream I/O in CSE143

What is a Stream?

- Think of a stream as an infinite sequence of characters
- Input streams allow **extraction**: get a character from the stream
- Output streams allow **insertion**: put a character into the stream

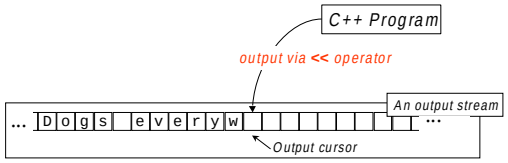
Input Streams

- **istream** and **ifstream** are types of input streams
  - Keyboard (**istream**), file (**ifstream**)



Output Streams

- **ostream** and **ofstream** are types of output streams
  - Screen (**ostream**), file (**ifstream**)



Well-Known Streams

- Global streams defined in **iostream.h** :
  - **cin**: standard input stream (usually keyboard)
  - **cout**: standard output stream (usually screen)
  - **cerr**: standard error stream
- Programs can open other streams to/from files and other devices:
  - **fstream.h**: for file input and output
  - **strstream.h**: for stream interface to strings

Stream Output Operation

- For output streams, << is the “put to” or "insertion" operator

```
#include <iostream.h>

int age = 17;
cout << "When I was " << age
     << " it was a very good year."
     << endl;
```

- **endl** is a special symbol: adds a newline

Stream Input Operation

- For input streams, **>>** is the “get from” or "extraction" operator

```
#include <iostream.h>
…
int x, ID;
char name[40];
cin >> x;
cin >> name >> ID;
```

- Can read multiple items on one line
- No pesky **&**’s as needed with **scanf** (how?)

How Stream Input Works

- Input stream characters are interpreted differently, depending on the data type:

```
#include <iostream.h>
…
int ID;
char ch, name[40];
cin >> ID;    // Tries to read an integer
cin >> ch;    // Tries to read a char;
              // skips whitespace
cin >> name;  // Reads character string;
              // skips leading whitespace,
              // stops at trailing whitespace
```


Preview of C++ Classes

- Streams are instances of C++ classes
- Like structs, class instances have fields (generally called members).
- Members can be data or functions (methods)
- We use the “.” syntax to access members.

```
x = aStudent.gpa;   // Access data field foo, like C
y = aStudent.register( SUMMER );
                    // Call aStudent’s register method,
                    // put result in y
```

- (much) more on this later

Other Stream Operations

- Sometimes want to read from a stream **without** ignoring whitespace (spaces, tabs, newlines)

```
char c;
cin.get( c ); // c gets next char from cin
```

- **get** also returns a status value (*e.g.*, for end-of-file)
- See Appendix C and Lippman for details

File Stream Example

```
#include <iostream.h>
#include <fstream.h>

void main() {
      ifstream inFile( "input.txt" );  // open input
      ofstream outfile( "output.txt" ); // open output
      char ch;

      // should test for successful opens here..

      while( inFile.get(ch) ) {  // while more input
            outFile.put(ch);     // write it to output
      }

      inFile.close();            // close the files
      outFile.close();
}
```


Stream Testing (I)

- We often want to test the status of a stream, to see if it has any more data.
- It's usually cleanest to use the return value of the input function ...

```
while( cin.get(ch) ) {
      // Continue until get fails
}
while( cin >> someInt ) {
      // Similar to above, but details are hidden
}
```


Stream Testing (II)

- It's also possible to use the eof() member function:

```
if( !someFile.eof() ) {
     doMoreWithFile( someFile );
}
```

- For keyboard input, special characters such as *Ctrl-Z (on most PCs) or Ctrl-D (on UNIX) signal end-of-file

*"pronounced "zed"

Other Useful Functions

- **putback** allows you to put the character you just extracted back into an input stream

```
cin.get( c );
if( c == '(' ) { cin.putback( c ); }
```

- **peek** allows you to look at the next char

```
c = cin.peek();
if( c != '(' ) { cin.get( c ); }
```


Stream Advantages

- Type Safety
  - No more dangers of mismatched %-specifiers

```
scanf( "%s", &someInt );      // oops!
scanf( "%d", someInt );       // doh!
```

- Readability
  - Easier to read than a sequence of %-spe

```
scanf( "%d%c%d", &i1, &c2, &i3 );
```

```
cin >> i1 >> c2 >> i3;
```