“Formatted” versus “Unformatted” Files (all files are formatted somehow)
Unix has NO BUILT IN INTERPRETATIONS of application file contents: Every file is a stream of 8 bit bytes.

“Formatted, ASCII, text” file usually means a file that (should) contain only bytes in the (printable ASCII) 32_{10} \text{ to } 126_{10} plus whitespace (8-13) range.
The formatted stream “get from” (\textgreater\textgreater) and “put to” (\textless\textless) operators convert between conventional printed data value forms and the representations in C++ variables as bits.
>> and << are **type-safe**: The type of C++ variable they are applied to controls what they do, so

- Input format errors are detected; target variable is unchanged when there’s a format error.
- Bits in each output variable are interpreted according to the variable’s C++ type.
- Type mismatch is avoided.

Format options can be set for individual streams using members of class `ios` (Strou. 21.4).

---

Situation where formatted I/O is not desirable:

1. Complicated, application specific rules for analyzing input: Writing a C++ preprocessor or compiler, for example. Input should be read character by character (or line by line) and processed by a scanner.

2. Bits in the file should be identical with bits processed in C++ memory.
Slide 4

Functions for line-by-line input (illustrated on MyF)

```cpp
char inbuf[BUFSIZE];
MyF.getline( inbuf, BUFSIZE );
MyF.getline( inbuf, BUFSIZE, 'Z' );
    // strange line terminator Z
MyF.get( inbuf, BUFSIZE );
    // LEAVES \n in the stream
cout << "Just read " << MyF.gcount() 
    " chars!" << endl;
```

---

Slide 5

Functions for character-by-character input

```cpp
char inbuf[BUFSIZE];
MyF.get( inbuf[i] );
Type of inbuf[i] is char
inbuf[i] = MyF.get();
```
How to read a MIPS memory image:

// make NWORDS = 256 for this Project 3
long unsigned int pMem =
    new long unsigned int[ NWORDS ];
MyF.read( pMem,
           NWORDS*sizeof(long unsigned int) );
if( MyF.gcount() !=
           NWORDS*sizeof(long unsigned int) )
   { // something’s wrong! }
istream::read( pointer, count ) takes a pointer
to the destination and a count of the maximum number
of bytes to read. There BETTER be at least count
bytes available where pointer points!

How to write a MIPS memory image:

// make NWORDS = 256 for this Project 3
long unsigned int pMem =
    new long unsigned int[ NWORDS ];
MyOutF.write( pMem,
              NWORDS*sizeof(long unsigned int) );
ostream::write( pointer, count ) takes a pointer
to the source array and a count of the number of bytes
to copy out and write. It is the programmer’s
responsibility to make sure the output is meaningful.