Library In A Week – Day 2

Revamping C++ I/O

Jeff Garland

jeff@crystalclearsoftware.com

Agenda

- Issues Presentation
  - Kevin, Thomas, Krishna
- Solutions
  - Boost format – Rob
  - Boost iostreams / asio – Jeff (maybe no time)
- Directions
- Tommorrow...
  - Iochain - Sebastian

Issue Summary - Performance

- Fixed cost for each extraction (shouldn't be required)
- Manipulators use function pointers which are costly
- Virtual function overhead
- std::stringstream is expensive
  - Allocations from underlying string

Design Principles

- Ideally separate formatting/parsing from buffering completely
  - Likely requires a do over
- Backward compatiblity
  - Perhaps a layer on top

Example

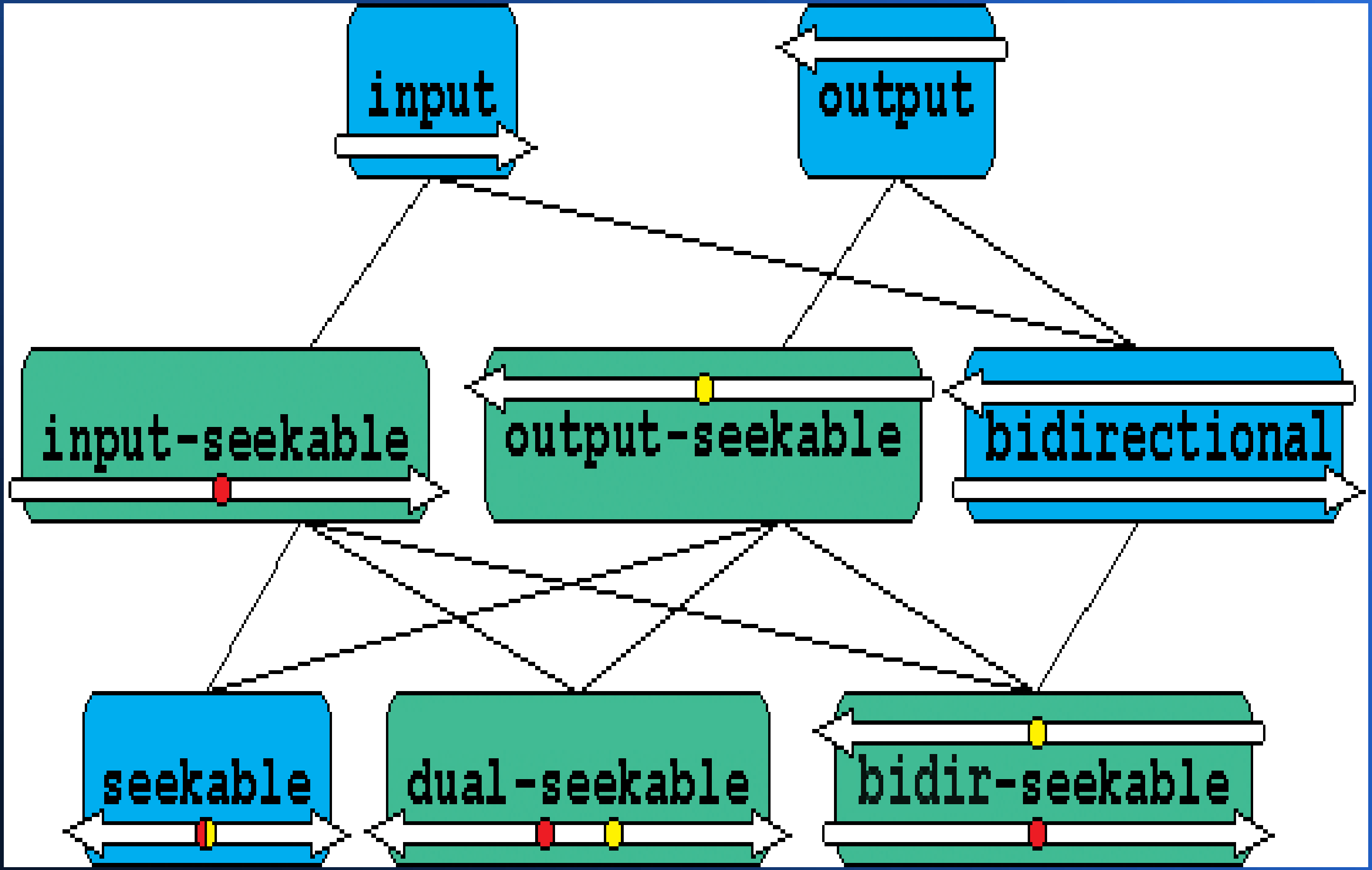
boost::iostreams device concepts

- Device - character type and category
- Exemplars
  - Source – read only device for input
    - size_type io::read(dev, s1, n)
    - std::pair<Ch*,Ch*> dev.input_sequence()
  - Sink – write only device for output
    - size_type io::write(dev, s2, n)
    - std::pair<Ch*,Ch*> dev.output_sequence()
  - Bi-Directional – read and write, 2 pointers
  - Seekable – read/write with repositioning

boost::iostreams filter concepts

- Filter
  - Filter operates on the character sequence or sequences controlled by a Device
  - Must have char_type of device
- Exemplars
  - InputFilter
    - Tr::int_type f.get(d)
    - size_type f.read(d, s, n)
  - OutputFilter
    - bool f.put(d, c)

boost::iostreams modes


input
output
input-seekable
output-seekable
bidirectional
seekable
dual-seekable
bidir-seekable