Today

- Stream insertion and extraction
- Basic Sequence class

Assignments

- HW-1 due
- HW-2 out
- Quiz 2 on Thursday

Note on terminology:

- "Overload": Same function name, different signature
- "Override": Same signature in base and subclass
Stream insertion and extraction operators

Stream insertion (<<) and extraction (>>) must be non-member functions!

- e.g., for: `cout << r1;` or `cin >> r1;`

Example

```cpp
class Rectangle
{
public:
...

friend std::ostream& operator<<(std::ostream& out, const Rectangle& rhs);

friend std::istream& operator>>(std::istream& in, Rectangle& rhs);
};

// implementation ...

std::ostream& operator<<(std::ostream& out, const Rectangle& rhs)
{
    out << rhs.length << " " << rhs.width;
    return out;
}

std::istream& operator>>(std::istream& in, Rectangle& rhs)
{
    in >> rhs.length;
    in >> rhs.width;
    return in;
}

Q: Why do we return the streams?

Q: Explain the signatures (const, references, return types)
A Basic “Sequence” ADT

A **Sequence** is an **ordered collection** of items

- order is based on position in the collection
- duplicate values are allowed (as opposed to a set)

A bare-bones **Sequence** class where items are stored as integers

- *Note this class has problems we are going to fix as we go ...*

```cpp
class Sequence {
public:
    int size() const; // returns length
    bool empty() const; // true if empty
    int& operator[](int index); // set value at index
    const int& operator[](int index) const; // get value at index
    void insert(int elem, int index); // grow at index
    void erase(int index); // shrink at index
    bool contains(int elem) const; // check membership
};
```

Examples:

```cpp
Sequence s;
s.insert(10, 0);
s.insert(20, 1);
cout << s[0] << " " << s[1] << endl; // prints: 10 20
s[0] = 15;
s.erase(1);
cout << s.size() << " " << s[0] << endl; // prints: 1 15
```

Q: How should we handle invalid indexes?

- Option 1: do nothing
- Option 2: throw exception (checked indexes) ... our approach, more later
A linked list subclass implementation

One way to implement Sequence is using a linked-list data structure

- we’ll define linked-list version as a subclass of Sequence
- since there can be other ways to implement sequences as well
- note again that there are issues below, which we’ll fix as we go!

```cpp
class LinkedSeq : public Sequence
{
public:
    // constructors, destructors, copy/move operators
    ...

    // overriden from Sequence
    int size() const;
    bool empty() const;
    int& operator[](int index);
    const int& operator[](int index) const;
    void insert(int elem, int index);
    void erase(int index);
    bool contains(int elem) const;

private:
    struct Node {
        int value;
        Node* next;
    };
    Node* head;
    Node* tail;
    int length;
};
```

Q: Can you spot any issues in this implementation?