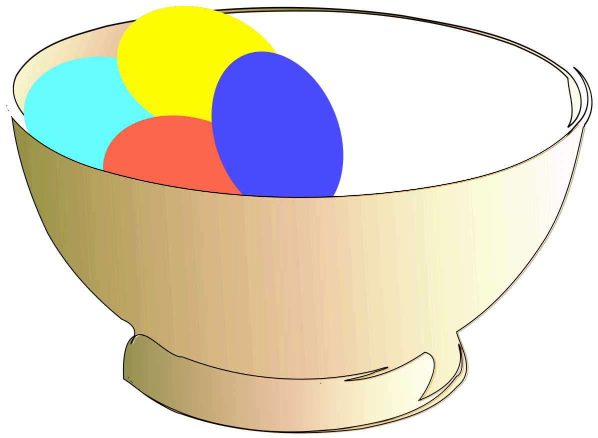
Review: Containers, Positions, and Iterators

What is a container?

Containers

Objects that can hold other objects/variables and more…



What is a position?

Position: stores the node reference, but privately!

```
class Position{
    public:
        E& element();
    private:
        Node* v;
    };
```

Node Structure

```
struct Node {
  Elem elem;
  Node* prev;
  Node* next;
};
```

Overload operator * to return element!

```
class Position{
    public:
        E& operator*();
    private:
        Node* v;
    };
```

```
E& Position::operator*{
    return v->elem;
};
```

```
Position p;
p = S.top();
E& elm = *p;
```

What is an Iterator?

Overload -- and ++ for Position

```
Position& Position::operator++{
    v=v−>next;
    return *this;
};
```

Iterator Class

```
class Iterator {
public:
    Elem& operator*();
    bool operator==(const Iterator& p) const;
    bool operator!=(const Iterator& p) const;
    Iterator& operator++();
    Iterator&  operator−−();
    friend class NodeList;
private:
    Node* v; Iterator(Node* u);
};
```

List Container

```
typedef int Elem;                                      // list base element type
class NodeList {                                       // node-based list
private:
  // insert Node declaration here. . .
public:
  // insert Iterator declaration here. . .
public:
  NodeList();                                          // default constructor
  int size() const;                                    // list size
  bool empty() const;                                  // is the list empty?
  Iterator begin() const;                              // beginning position
  Iterator end() const;                                // (just beyond) last position
  void insertFront(const Elem& e);                     // insert at front
  void insertBack(const Elem& e);                      // insert at rear
  void insert(const Iterator& p, const Elem& e); // insert e before p
  void eraseFront();                                   // remove first
  void eraseBack();                                    // remove last
  void erase(const Iterator& p);                       // remove p
  // housekeeping functions omitted. . .
private:                                               // data members
  int     n;                                           // number of items
  Node*   header;                                      // head-of-list sentinel
  Node*   trailer;                                     // tail-of-list sentinel
};
```

++ Overloading

```
Iterator& Iterator::operator++{
        v = v->next;
        return *this;
};
```

Erase with Iterator

```
void NodeList::erase(const Iterator& p) {
  Node* v = p.v;
  Node* w = v->next;
  Node* u = v->prev;
  u->next = w; w->prev = u;
  delete v;
  n--;
}
```

The “Position” of a Node

```
class Position <E>{
    public:
        E& operator*();
        Position parent () const;
        PositionList children () const;
        bool isRoot() const;
        bool isExternal() const;
    private:
        …
}
```