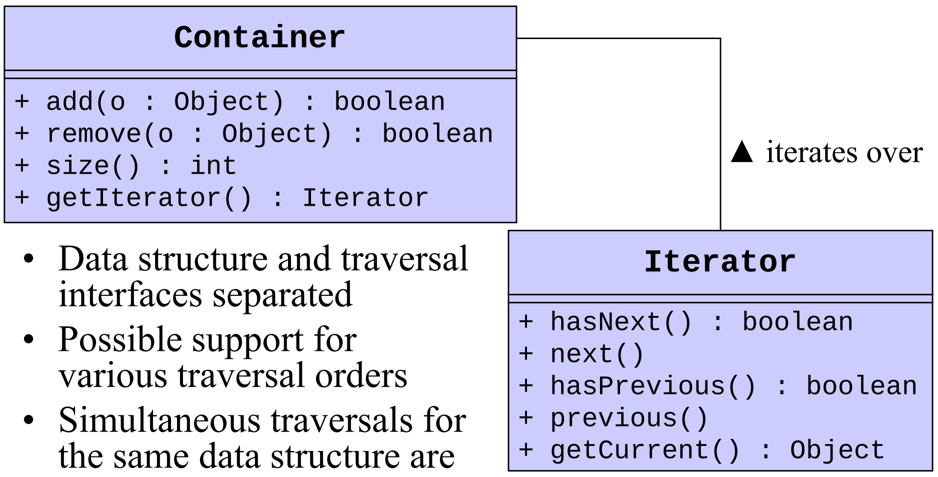
Iterator Design Pattern



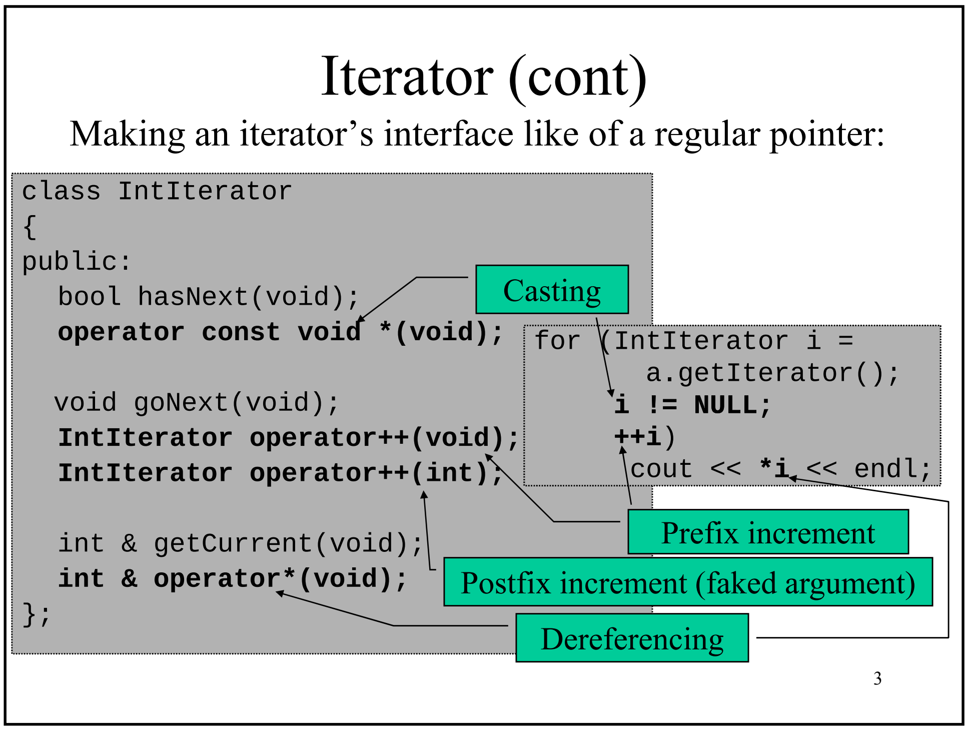
- Data structure and traversal interfaces separated
- Possible support for various traversal orders
- Simultaneous traversals for the same data structure are supported

Iterator

```
class IntArray
{
public:
   IntIterator getIterator(void);
};

class IntIterator
{
   friend IntIterator IntArray::getIterator(void);
private:
   int * const pnData;
   const int  nSize;
         int  nCurrent;
   IntIterator(int *const pnDataInit,int nSizeInit);
};
```

The iterator instantiation is a container's responsibility



Iterator (cont)

Making an iterator’s interface like of a regular pointer:

```
class IntIterator
{
public:
   bool hasNext(void);
   operator const void *(void);

   void goNext(void);
   IntIterator operator++(void);
   IntIterator operator++(int);

   int & getCurrent(void);
   int & operator*(void);
};
```

```
for (IntIterator i =
       a.getIterator();
     i != NULL;
     ++i)
    cout << *i << endl;
```

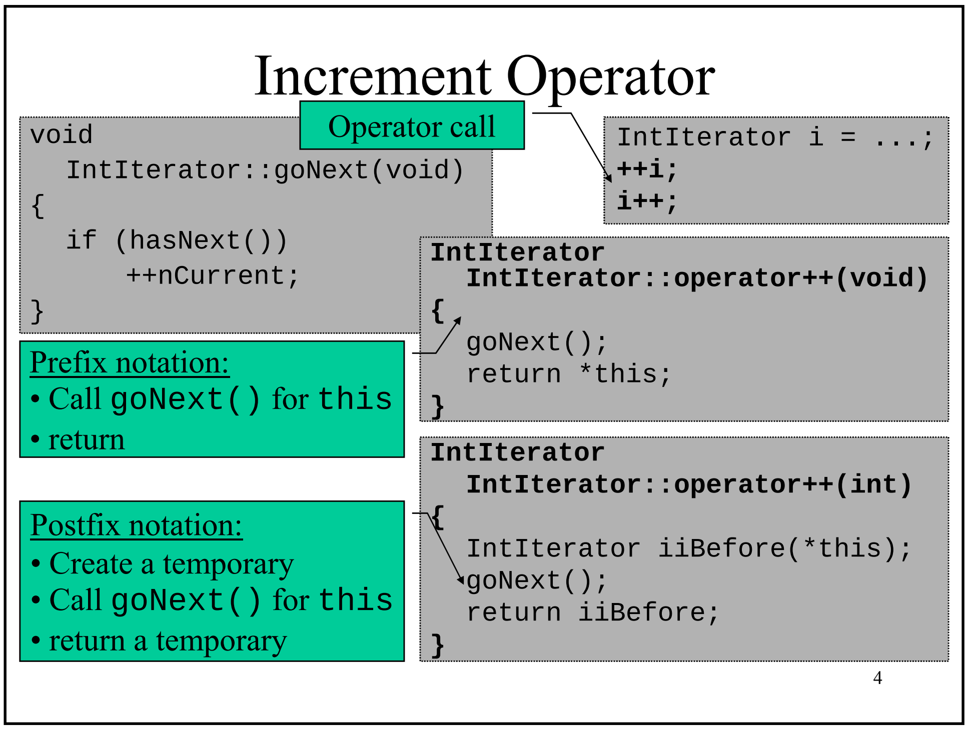
Casting

Prefix increment

Postfix increment (faked argument)

Dereferencing

3



Increment Operator

```
void
   IntIterator::goNext(void)
{
   if (hasNext())
        ++nCurrent;
}
```

Operator call

```
IntIterator i = ...;
++i;
i++;
```

```
IntIterator
   IntIterator::operator++(void)
{
   goNext();
   return *this;
}
```

Prefix notation:
- Call `goNext()` for `this`
- return

```
IntIterator
   IntIterator::operator++(int)
{
   IntIterator iiBefore(*this);
   goNext();
   return iiBefore;
}
```

Postfix notation:
- Create a temporary
- Call `goNext()` for `this`
- return a temporary

4

Dereferencing Operator

```
IntIterator i = ...;
int nA = *i;
```

Operator call

```
int & IntIterator::getCurrent(void)
{ return pnData[nCurrent]; }
```

```
int & IntIterator::operator*(void)
{ return getCurrent(); }
```

5

Casting Operator

```
IntIterator i = ...;
bool bHasNext = (i == NULL);
```

Operator call

```
bool IntIterator::hasNext(void)
{ return nCurrent < nLast; }
```

```
IntIterator::operator const void *(void)
{
  return
    hasNext() ? pnData + nCurrent : NULL;
}
```

6

Casting Operator (cont)

$$\begin{tikzcd}\text{Current Type} \arrow[d, bend right, "\textbf{Casting operator}"'] \\ \text{Other Type} \arrow[u, bend right, "\textbf{Conversion constructor}"']\end{tikzcd}$$

```
class Rational
{
public:
  operator const double(void);
private:
  int nNominator, nDenominator
};
```

```
Rational::operator const double(void)
{
  return
    ((double)nNominator)/nDenominator;
}
```

```
double dPi = 3.14;
Rational r(2,3);
cout << (dPi+r);
```

Will be casted to `double`

7

Function Call Operator

Functor is an object that encapsulates a function call

```
class CounterFunctor
{
public:
  CounterFunctor(void);
  int getNext(void);
  int getLast(void);
  int operator()(void);
private:
  int nLast;
};

int CounterFunctor::getNext(void)
{ return ++nLast; }
int CounterFunctor::operator()(void)
{ return getNext(); }
```

```
CounterFunctor nextInt;

for (int nCurrent;
  nCurrent<10;
  ++nCurrent)
  cout << nextInt()
       << endl;
```

Usage example

8